

FIG. 1

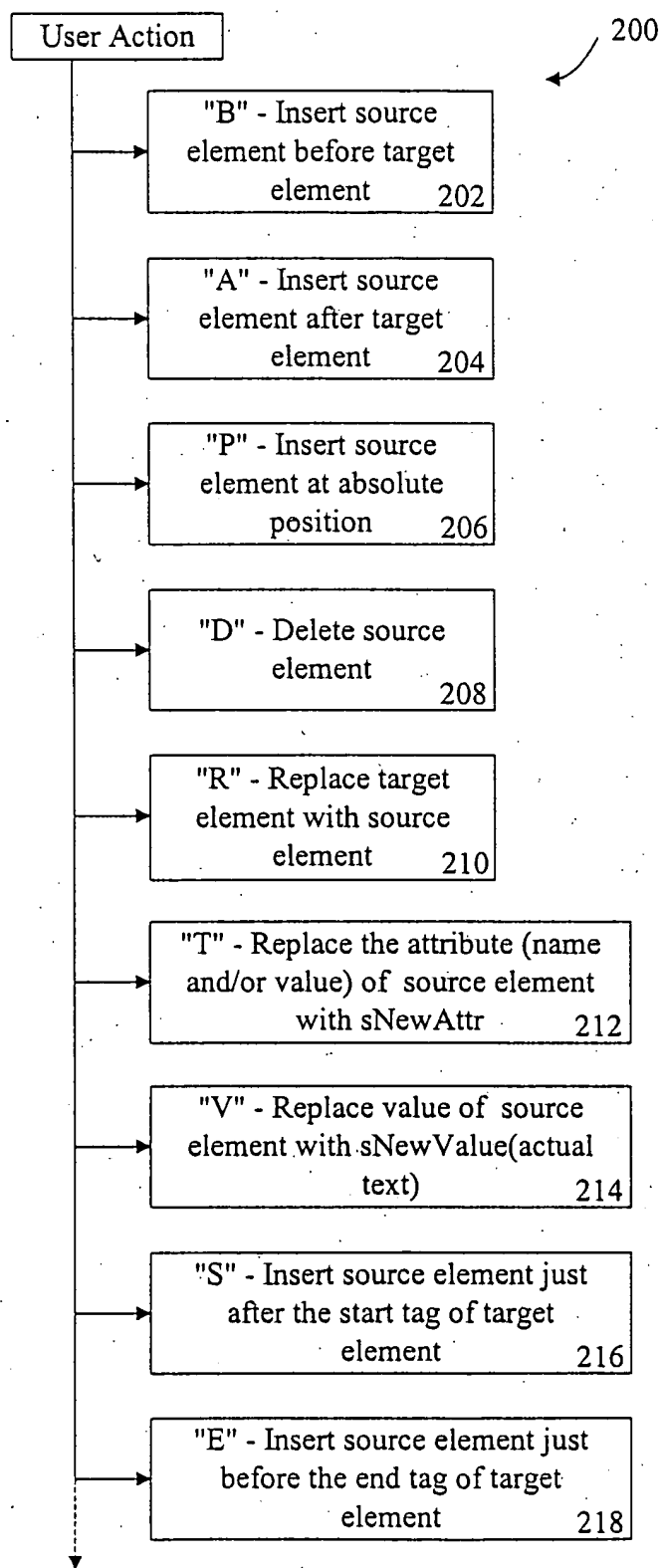


FIG. 2

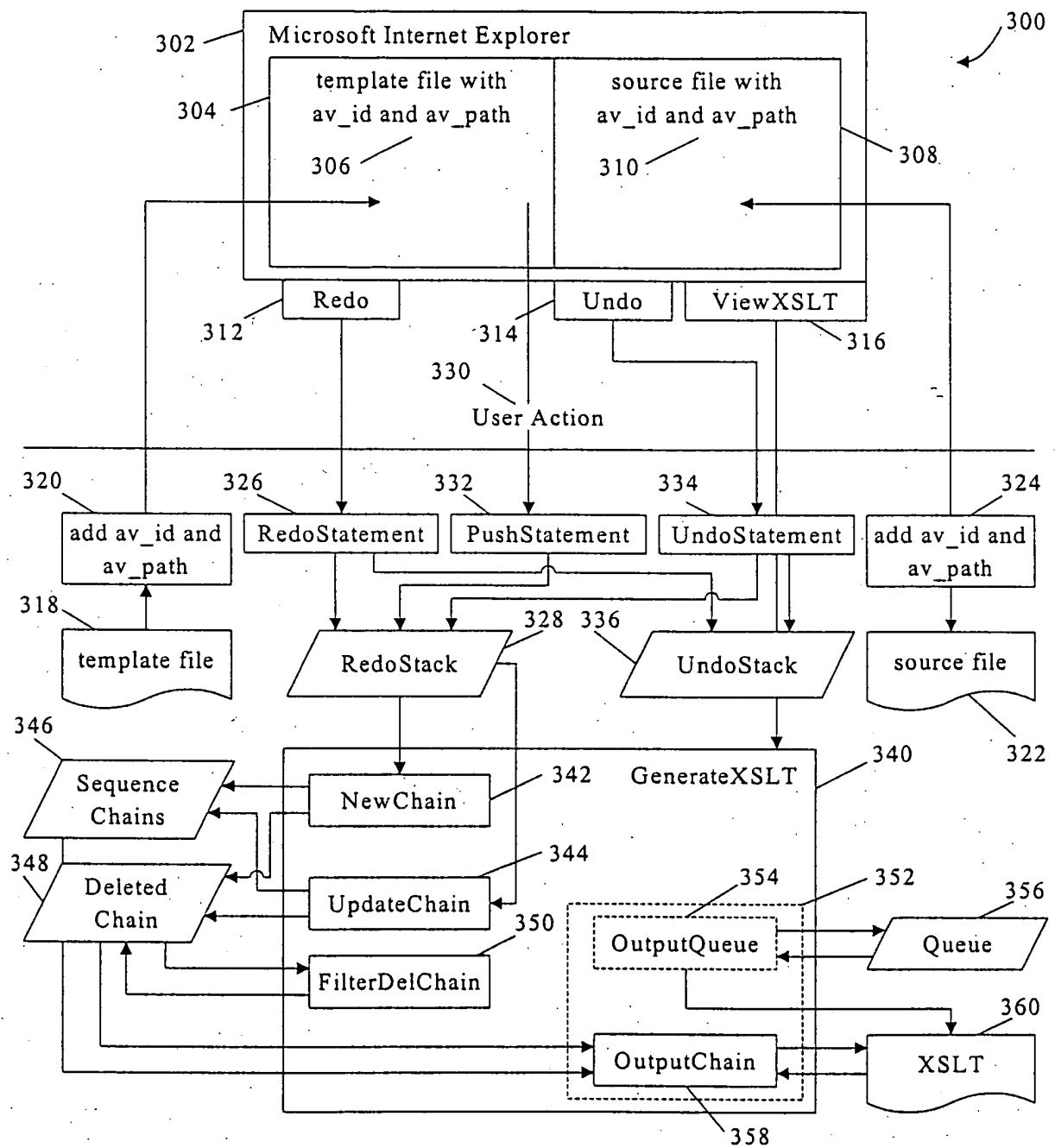


FIG. 3A

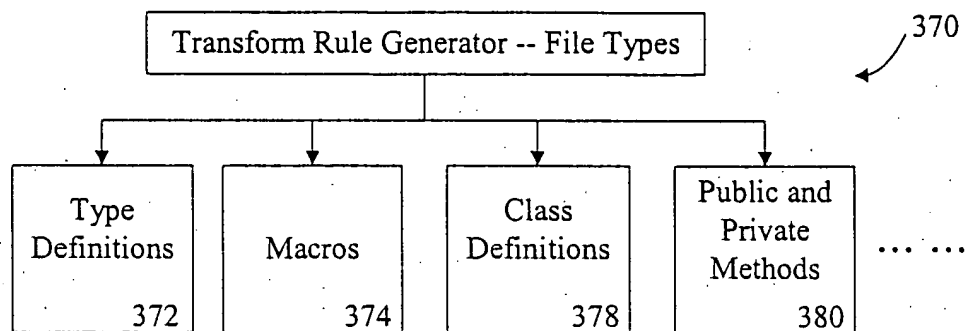


FIG. 3B

```

                                TagId
typedef struct _TagId{
    int        iPage;        // page /deck number
    int        iCard;        // card number for WML only
    int        iFamilyId;
    DOMString  sFrame;        //sample: i1.3.2
    DOMString  sNewTag;
    DOMString  sId;
    DOMString  sName;
    DOMString  sPath;        //sample: tmp/html[1]/body[1]
    DOMString  sAbsPos;
    bool       bIsChanged;
    bool       bIsAbsPosOrg;
    char       cAbsPos;
    int        x, y;
}TagId;

```

FIG. 4A

```

                                Statement
type defstruct _Statement{
    bool       bNewAction;
    char       cAction;
    struct _TagId  sourceEle;
    struct _TagId  targetEle;
    DOMString  *psNewAttrName;
    DOMString  *psNewAttrValue;
    int        iNumOfAttr;
    DOMString  sNewText;
    struct _Statement*pPrev;
    struct _Statement*pNext;
}Statement;

```

FIG. 4B

```

                                Stack
typedef struct _Stack{
    Statement  *pFirstStatement,
    Statement  *pLastStatement;
}Stack;

```

FIG. 4C



Deleted Chains



FIG. 4G

Sequence Chain



FIG. 4H

Sequence Chain with Child Chain

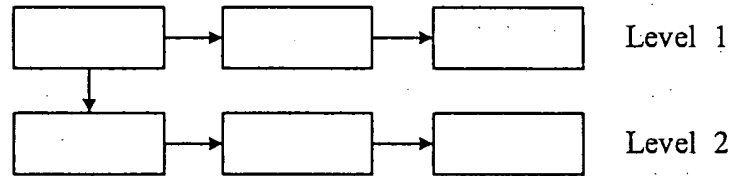


FIG. 4I

Card

```

typedef struct _Card{
    int          iCard;
    int          iEntry;
    struct _Chain *pFirstChain;
    struct _Chain *pLastChain;
    struct _Chain DelChain;
    struct _Unit  *pFirstUnit;
    struct _Unit  *pLastUnit;
    struct _Card *pPrev;
    struct _Card *pNext;
}Card;
  
```

FIG. 4J

Page  
typedef struct \_Page{  
    int           iPage;  
    struct \_Card \*pFirstCard;  
    struct \_Card \*pLastCard;  
    struct \_Var   \*pRootTmpVar;  
    struct \_Var   \*pRootSrcVar;  
    struct \_Page \*pPrev;  
    struct \_Page \*pNext;  
}Page;

FIG. 4K



```

Attr
typedef struct _Attr{
    DOMString    sAttrName;
    DOMString    sAttrValue;
    struct _Attr *pPrev;
    struct _Attr *pNext;
}Attr;

```

FIG. 4L

```

Unit
typedef struct _Unit{
    struct _Element *pElement;
    struct _Unit *pPrev;
    struct _Unit *pNext;
}Unit;

```

FIG. 4M

```

ElementInfo
typedef struct _ElementInfo{
    bool          bSourceEleIsLocated;
    bool          bTargetEleIsLocated;
    struct _TagId sourceEle;
    struct _TagId targetEle;
    struct _Element *pSourceElement;
    struct _Element *pTargetElement;
    struct _Chain *pChainForSourceEle;
    struct _Chain *pChainForTargetEle;
}ElementInfo;

```

FIG. 4N

```

Var
typedef struct _Var{
    int      iMaxFrame;
    int      iMaxIFrame;
    DOMString sFrame;
    bool      bToOutput;
    struct _Var *pPrev;
    struct _Var *pNext;
    struct _Var *pFirstFrame;
    struct _Var *pLastFrame;
    struct _Var *pFirstIFrame;
    struct _Var *pLastIFrame;
} Var;

```

FIG. 4O

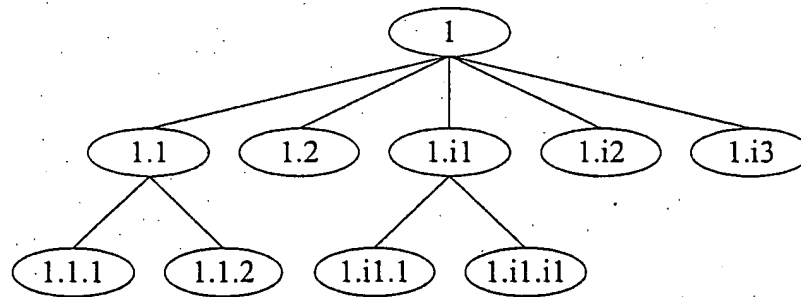


FIG. 4P

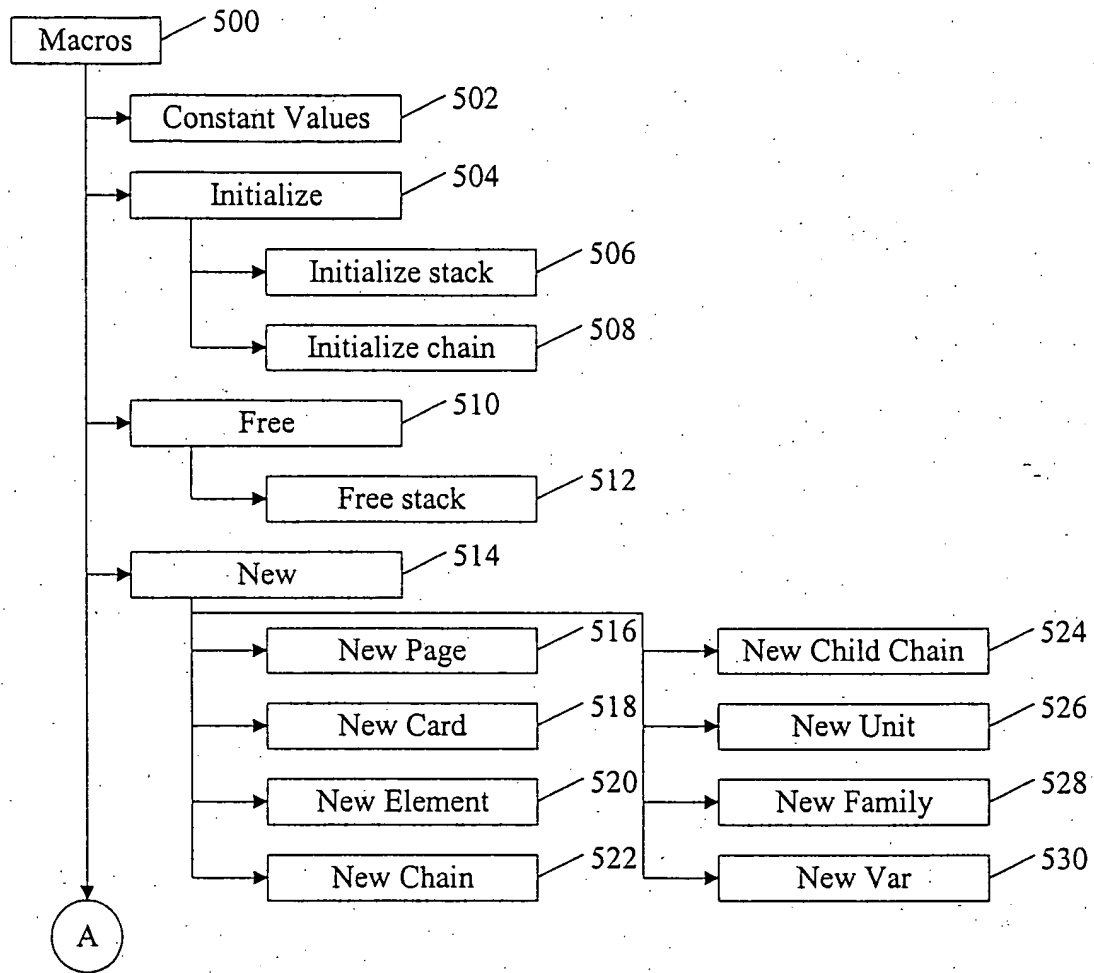


FIG. 5A

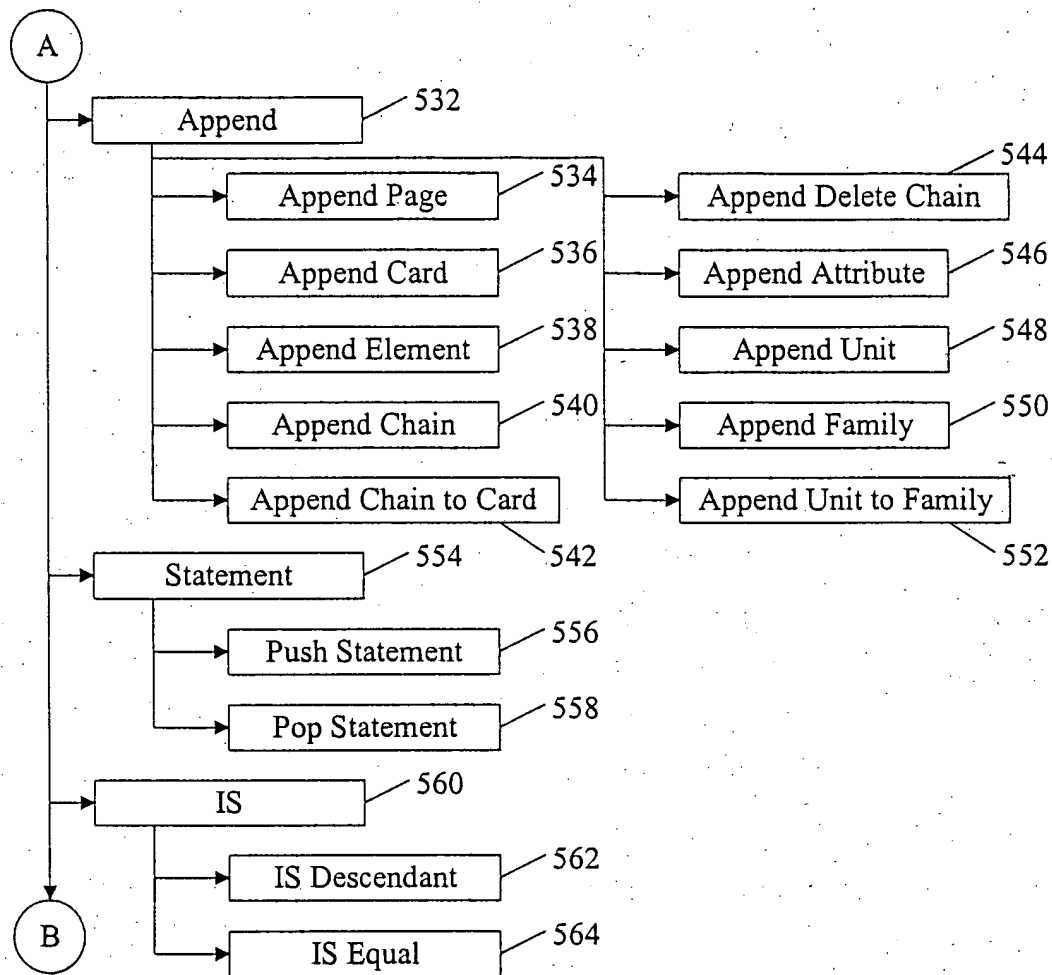


FIG. 5B

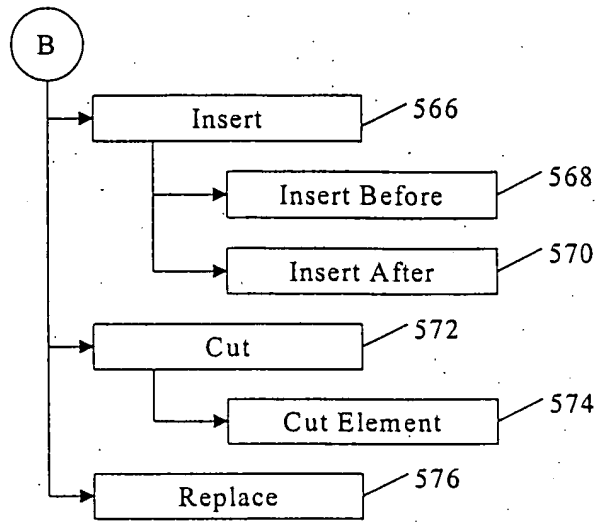


FIG. 5C

Constructor

```
m_pFirstPage      = NULL;
m_pLastPage       = NULL;
m_pFirstChain     = NULL;
m_pLastChain      = NULL;
m_pFirstDelChain  = NULL;
m_pLastDelChain   = NULL;

m_bCanRedo = m_bCanUndo = false;
INIT_STACK(m_redoStack);
INIT_STACK(m_undoStack);

NEW_CHILD_CHAIN(m_pDelChain);
```

FIG. 6A

Deconstructor

```
FREE_STACK(m_redoStack);
FREE_STACK(m_undoStack);

for(pPage = m_pFirstPage; pPage != NULL; ){
    for(pCard = pPage->pFirstCard; pCard != NULL; ){
        for(pChain = pCard->pFirstChain; pChain != NULL; ){
            pChain = DeleteChain(pChain, pCard, FROM_CARD);
        }
        FREE(pPage->pFirstCard, pCard);
    }
    FREE(m_pFirstPage, pPage);
}

for(pChain = m_pFirstDelChain; pChain != NULL; )
    pChain = DeleteChain(pChain, NULL, FROM_M_PFIRSTDELCHAIN);
```

FIG. 6B

Statement Methods: B, A, R, S, E

```
//set statement
pStatement = new Statement;
pStatement->bNewAction = bNewAction;
pStatement->cAction = cAction;
pStatement->sourceEle = sourceEle;
pStatement->targetEle = targetEle;
pStatement->pPrev = 0;
pStatement->pNext = 0;

//Push into redoStack
PUSH_STATEMENT(redoStack, pStatement);

//set undo/redo
m_bCanUndo = true;

return;
```

FIG. 7A

```
Statement Methods: P, D
//set statement
pStatement = new Statement;
pStatement->bNewAction = bNewAction;
pStatement->cAction = cAction;
pStatement->sourceEle = sourceEle;
pStatement->targetEle = sourceEle;
pStatement->pPrev = 0;
pStatement->pNext = 0;

//Push into redoStack
PUSH_STATEMENT(redoStack, pStatement);

//set undo/redo
m_bCanUndo = true;

return;
```

FIG. 7B



Statement Methods: T

```
//set statement
pStatement = new Statement;
pStatement->bNewAction = bNewAction;
pStatement->cAction = 'T';
pStatement->sourceEle = sourceEle;
pStatement->targetEle = sourceEle;
pStatement->iNumOfAttr = iNumOfAttr;
pStatement->pPrev = 0;
pStatement->pNext = 0;

//allocate memory for psNewAttrName and psNewAttrValue of pStatement
pStatement->psNewAttrName = new DOMString[iNumOfAttr];
pStatement->psNewAttrValue = new DOMString[iNumOfAttr];

//set psNewAttrName and psNewAttrValue of pStatement
for(i=0;i<iNumOfAttr;i++){
    (pStatement->psNewAttrName)[i] = psNewAttrName[i];
    (pStatement->psNewAttrValue)[i] = psNewAttrValue[i];
}

//Push into redoStack
PUSH_STATEMENT(m_redoStack, pStatement);

//set undo/redo
m_bCanUndo = true;
return;
```

FIG. 7C

Statement Methods: V

```
//set statement
pStatement = new Statement;
pStatement->bNewAction = bNewAction;
pStatement->cAction = 'V';
pStatement->sourceEle = sourceEle;
pStatement->targetEle = sourceEle;
pStatement->sNewText = sNewText;
pStatement->pPrev = 0;
pStatement->pNext = 0;

//Push into redoStack
PUSH_STATEMENT(m_redoStack, pStatement);

//set undo/redo
m_bCanUndo = true;

return;
```

FIG. 7D

UndoStatement

```
//pop out from redoStack
POP_STATEMENT(m_redoStack, pStatement);

//push into undoStack
PUSH_STATEMENT(m_undoStack, pStatement);

//set redo/undo
m_bCanRedo = true;
if(m_redoStack.pFirstStatement == NULL)
    m_iCanUndo = false;
return;
```

FIG. 8A

```
RedoStatement
//pop out from undoStack
POP_STATEMENT(m_undoStack, pStatement);

//push into redoStack
PUSH_STATEMENT(m_redoStack, pStatement);

//set redo/undo
m_bCanUndo = true;
if(m_undoStack.pFirstStatement == NULL)
    m_iCanRedo = false;
return;
```

FIG. 8B

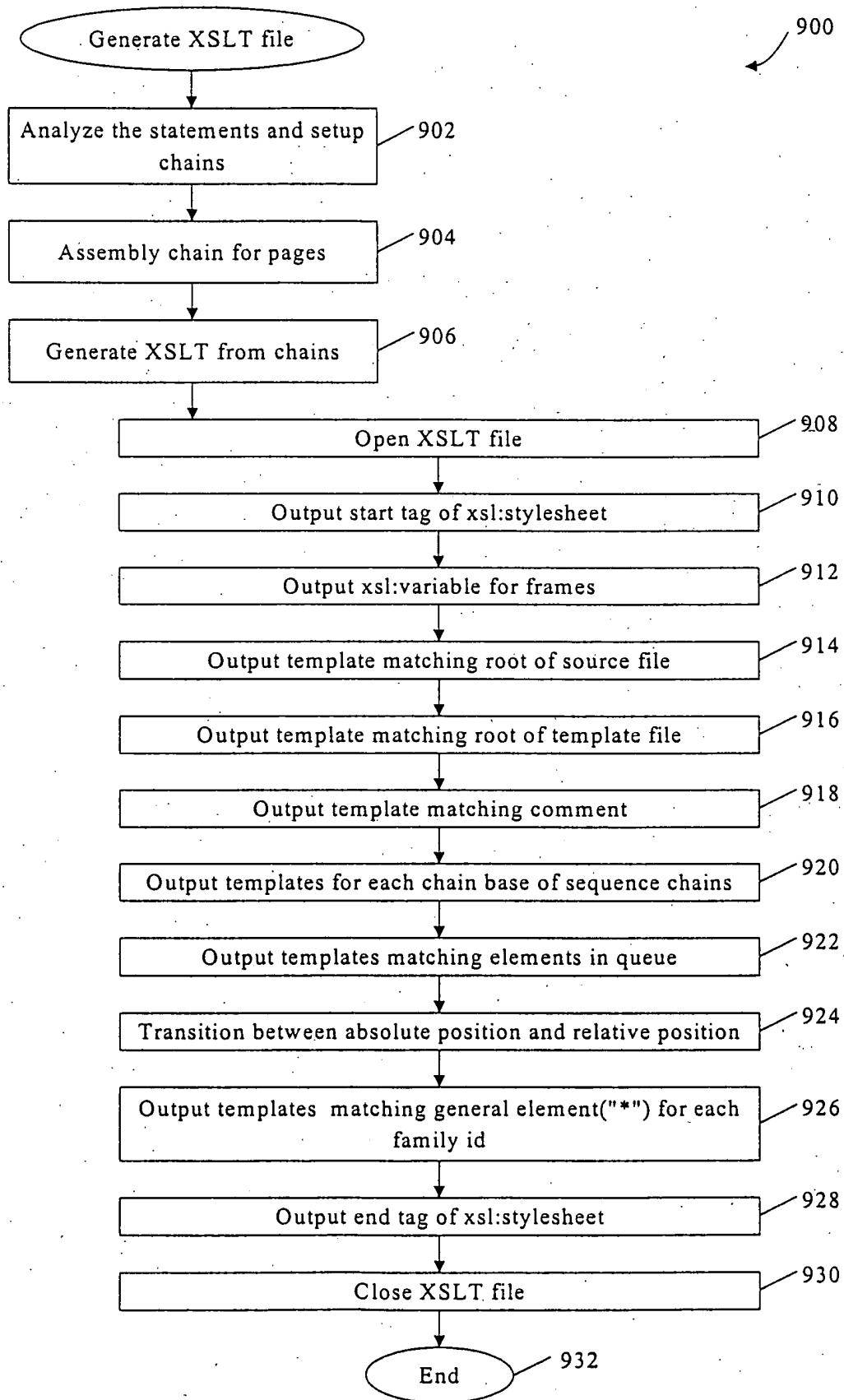


FIG. 9A

GenerateXSLT: Step 1

```
//step 1.1: loop from the first statement to the last statement to assembly sequence chains and
deleted chains
pCurrStatement = m_redoStack.pFirstStatement;
while(pCurrStatement != NULL){
    //step 1.1.1: assembly elementInfo
    elementInfo.sourceEle = pCurrStatement->sourceEle;
    elementInfo.targetEle = pCurrStatement->targetEle;
    elementInfo.pSourceElement = NULL;
    elementInfo.pChainForSourceEle = NULL;
    elementInfo.pTargetElement = NULL;
    elementInfo.pChainForTargetEle = NULL;
    //iPage equals 0 means that the element is from source page
    if(elementInfo.sourceEle.iPage == 0)
        elementInfo.bSourceEleIsLocated = true;
    else
        elementInfo.bTargetEleIsLocated = false;
    //targetEle's iPage is impossible to be = 0
    elementInfo.bTargetEleIsLocated = false;

    //step 1.1.2: Check whether sourceEle and targetEle are in sequence chains and call methods.
    LocateElement(m_pFirstChain, &elementInfo);

    //if targetEle is not in any sequence chain, call NewChain, otherwise call UpdateChain
    if(elementInfo.pTargetElement == NULL)
        NewChain(&elementInfo, pCurrStatement);
    else{
        UpdateChain(&elementInfo, pCurrStatement);
    }
    //step 1.1.3: Transit to the next statement
    pCurrStatement = pCurrStatement->pNext;
} //end of while

//step 1.2: Filter the deleted chain
FilterDelChain(pPage);
```

FIG. 9B

GenerateXSLT: Step 3.4 – 3.5

**Step 3.4 Output template matching root of source page**

**Format**

```
<xsl:template match="/">
  <xsl:apply-templates select="$tmp/*" mode="tmp_root_output"/>
</xsl:template>
```

**Step 3.5 Output template matching root of template page**

**Format**

```
<xsl:template match="*" mode="tmp_root_output">
  <xsl:copy>
    <xsl:for-each select="@*"><xsl:copy/></xsl:for-each>
    <xsl:apply-templates select="//comment()"/>
    <xsl:apply-templates select="*|text()" mode="tmp_test_# "/>
    .....
  </xsl:copy>
</xsl:template>
```

FIG. 9C

GenerateXSLT: Step 3.6 – 3.7

**Step 3.6 Output template matching comment**

**Step 3.7 Output templates for each chain base of sequence chains**

**Format**

```
<xsl:template match="chain base's path" mode="XY_base_##">
  <xsl:apply-templates select="the path of the Element in the chain" mode="xy_output_#"/>
  ... ..
</xsl:template>
```

FIG. 9D

GenerateXSLT: Step 3.7

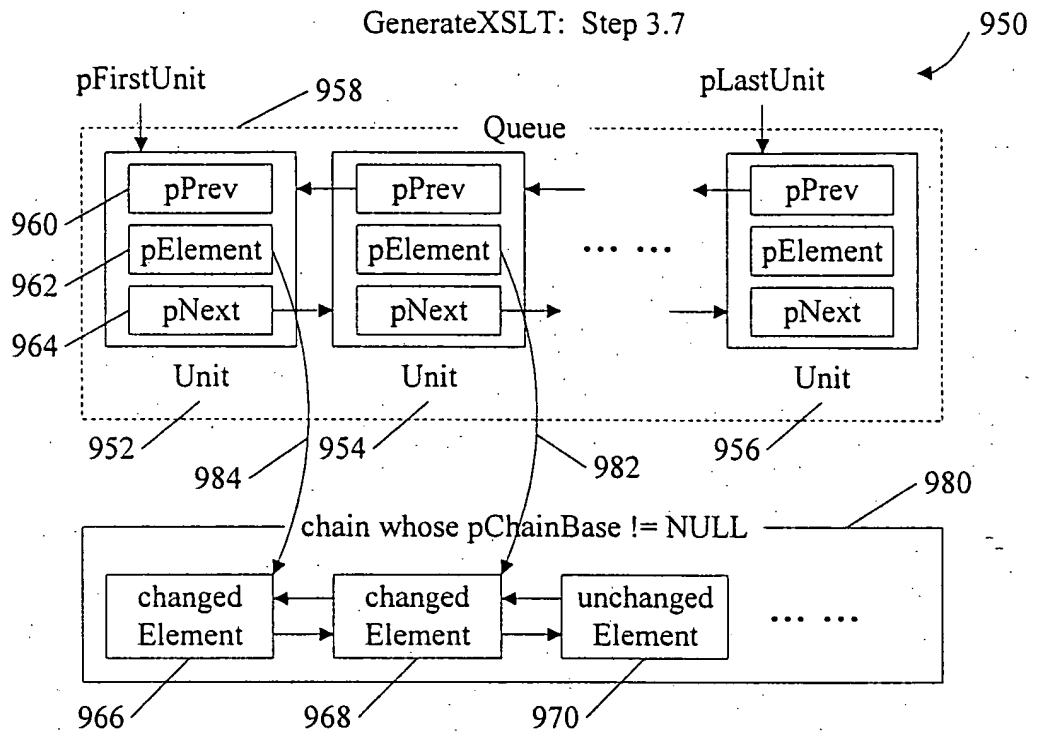
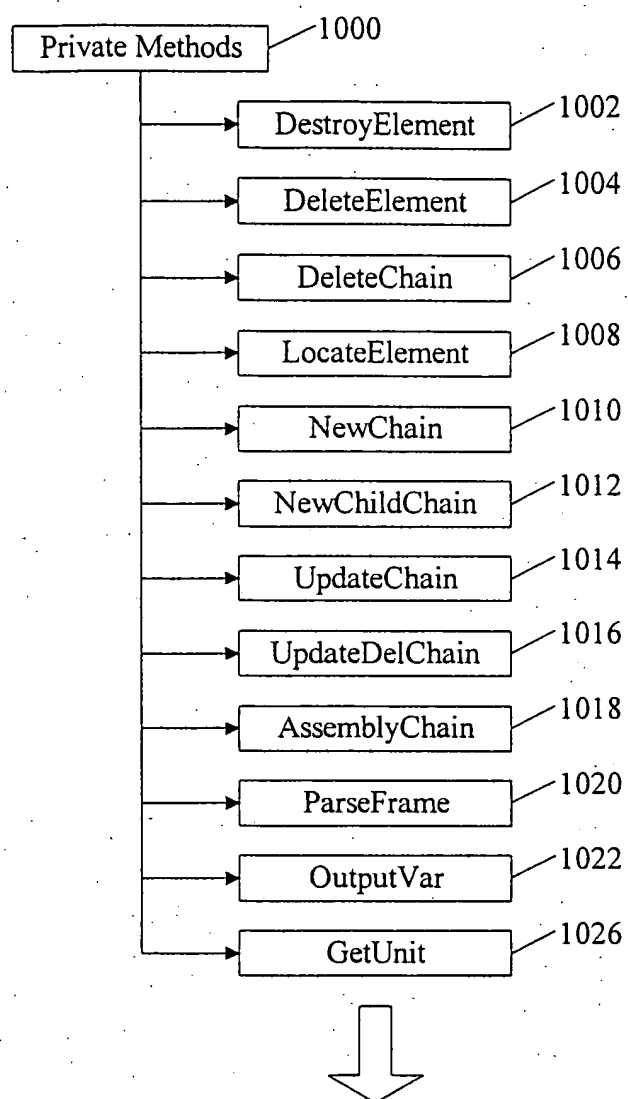


FIG. 9E





```

DestroyElement
if(pElement->pChildChain) DeleteChain(pElement->pChildChain);
free(pElement);

```

FIG. 11A

```

DeleteElement

Step 1. Cut the Element from the chain.
pNext = pElement->pNext;
CUT(pChain->pFirstElement, pChain->pLastElement, pElement);
Step 2. Destroy it.
DestroyElement(pElement);
Step 3. Finally return the pointer to the next Element.
return pNext;

```

FIG. 11B

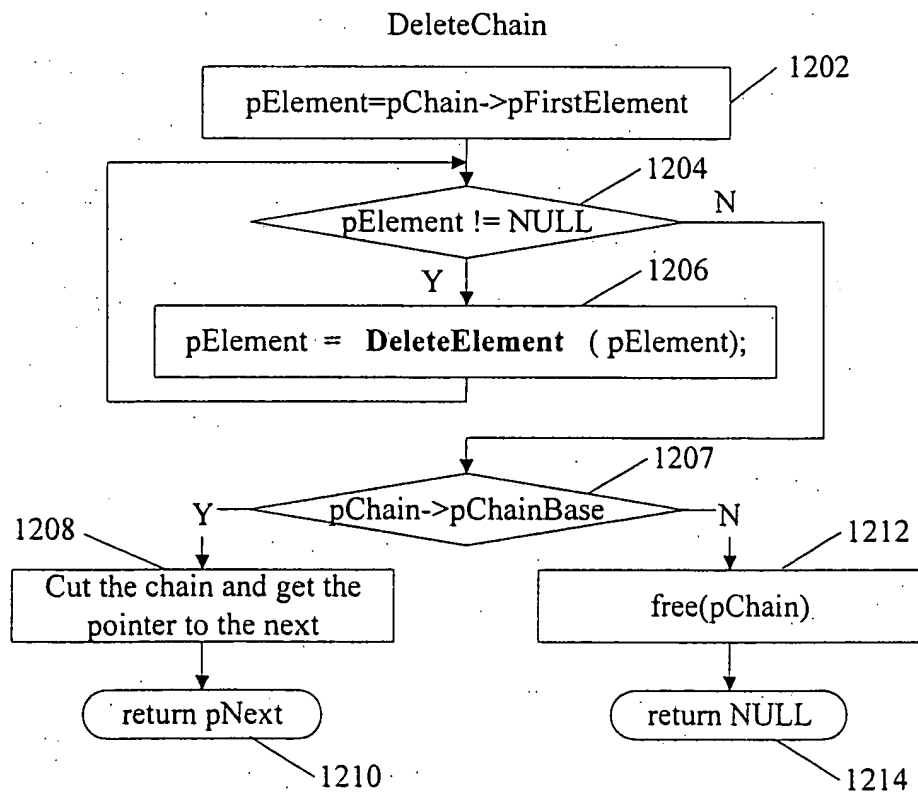


FIG. 12A

#### DeleteChain

```
pElement = pChain->pFirstElement;
//delete all Elements in this chain
while(pElement){
    pElement = DeleteElement(pChain, pElement);//return the next Element.
}
//delete the chain
if(pChain->pChainBase == NULL){//it is a child chain or Deleted Chain
    free(pChain);
    return NULL;
}
else{
    free(pChain->pChainBase);
    pNext = pChain->pNext;
    switch(mode){
        case FROM_M_PFIRSTCHAIN:
            CUT(m_pFirstChain, m_pLastChain, pChain);
            break;
        case FROM_CARD:
            CUT(pCard->pFirstChain, pCard->pLastChain, pChain);
            break;
        case FROM_M_PFIRSTDELCHAIN:
            CUT(m_pFirstDelChain, m_pDelLastChain, pChain);
            break;
    }
    free(pChain);
    return pNext;//MAY be NULL
}
```

FIG. 12B

# LocateElement(1)

```

sourceEle = pElementInfo->sourceEle;
targetEle = pElementInfo->targetEle;
bSourceEleIsLocated = pElementInfo->bSourceEleIsLocated;
bTargetEleIsLocated = pElementInfo->bTargetEleIsLocated;
//loop for all sequence chains
for(pRefChain = pChain; pRefChain != NULL; pRefChain = pRefChain->pNext){
    //set bToLocateSourceEle and bToLocateTargetEle
    if(pRefChain->pChainBase){
        if(bTargetEleIsLocated == false &&
           pRefChain->pChainBase->Ele.iPage == targetEle.iPage &&
           pRefChain->pChainBase->Ele.iCard == targetEle.iCard)
            bToLocateTargetEle = true;
        else
            bToLocateTargetEle = false;

        if(bSourceEleIsLocated == false &&
           pRefChain->pChainBase->Ele.iPage == sourceEle.iPage &&
           pRefChain->pChainBase->Ele.iCard == sourceEle.iCard)
            bToLocateSourceEle = true;
        else
            bToLocateSourceEle = false;
    }
    else{
        bToLocateSourceEle = !bSourceEleIsLocated;
        bToLocateTargetEle = !bTargetEleIsLocated;
    }
    //if bToLocateSourceEle or bToLocateTargetEle is not true, search in this chain
    if(bToLocateSourceEle || bToLocateTargetEle){
        //loop for all Elements in this chain
        pRefElement = pRefChain->pFirstElement;
        for( ; pRefElement; pRefElement = pRefElement->pNext){
            if(bToLocateSourceEle){
                if(IS_EQUAL(pRefElement->Ele, sourceEle)){
                    pElementInfo->pSourceElement = pRefElement;
                    pElementInfo->pChainForSourceEle = pRefChain;
                    pElementInfo->bSourceEleIsLocated = true;
                    bSourceEleIsLocated = true;
                }
            }
        }
    }
}

```

FIG. 13A

# LocateElement(2)

```

    }
    if(bToLocateTargetEle){
        if(IS_EQUAL(pRefElement->Ele, targetEle)){
            pElementInfo->pTargetElement = pRefElement;
            pElementInfo->pChainForTargetEle = pRefChain;
            pElementInfo->bTargetEleIsLocated = true;
            bTargetEleIsLocated = true;
        }
    }

    //if bToLocateSourceEle or bToLocateTargetEle is not true and this Element
    //has child chain, recursively call to search in the child chain.
    if(pRefElement->pChildChain && (!bSourceEleIsLocated || !bTargetEleIsLocated))
        LocateElement(pRefElement->pChildChain, pElementInfo);
    //if both are found, return, otherwise transit to the next Element.
    if((pElementInfo->bSourceEleIsLocated) && (pElementInfo->bTargetEleIsLocated))
        return;
    }//end of loop for pRefElement
} //end of if(bToLocateSourceEle || bToLocateTargetEle)
} //end of loop for sequence chains
else{
    //impossible to be here.
}

```

FIG. 13B

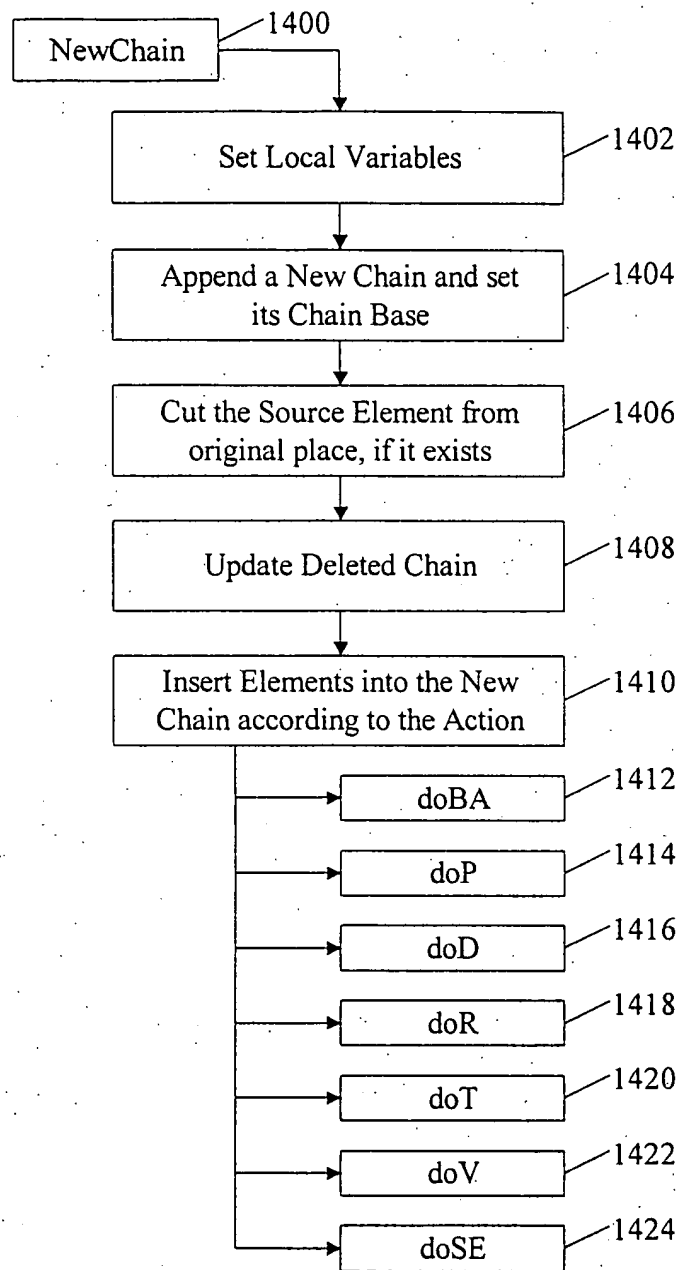


FIG. 14A

# NewChain(1)

## Step 1.: Set local variables

```
sourceEle = pStatement->sourceEle; targetEle = pStatement->targetEle;
cAction = pStatement->cAction;
pSourceElement = pElementInfo->pSourceElement;
pTargetElement = pElementInfo->pTargetElement;
```

## Step 2. Append a new chain and set its chain base.

```
if(cAction != 'D'){
    NEW_CHAIN(pChain, targetEle);
    APPEND_CHAIN(pChain);
}
```

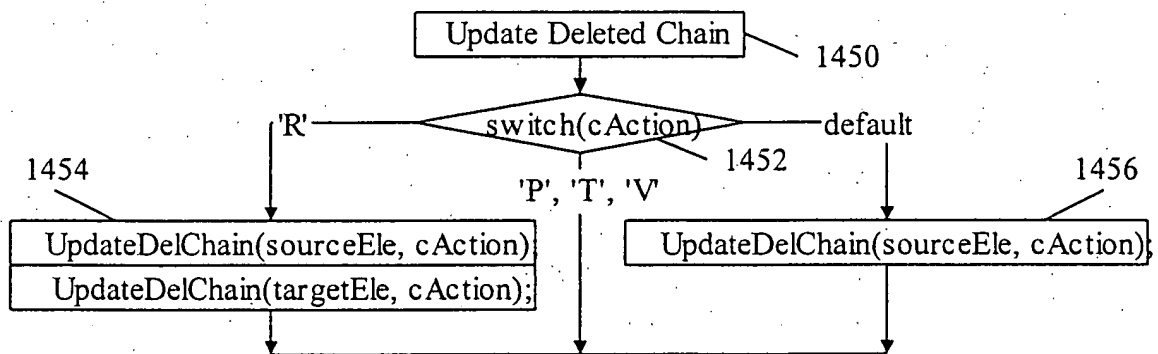
## Step 3. Cut the source Element from the original place if it exists

```
if(pSourceElement &&
    (cAction == 'B' || cAction == 'A' || cAction == 'S' || cAction == 'E' ||
    cAction == 'R' || cAction == 'D')){
    //Cut the Element from the original place.
    CUT(pElementInfo->pChainForSourceEle->pFirstElement,
        pElementInfo->pChainForSourceEle->pLastElement,
        pSourceElement);
}
```

//other actions are PTV

## Step 4. Update deleted chain

Flowchart



Pseudo code

```
switch(cAction){
    case 'R': UpdateDelChain(sourceEle, cAction);
              UpdateDelChain(targetEle, cAction); break;
    case 'P':
    case 'T':
    case 'V': break;
    default: UpdateDelChain(sourceEle, cAction); break; // 'B', 'A', 'S', 'E', 'D'
}
```

FIG. 14B

# NewChain(2)

**Step 5. Insert Elements into the new chain according to the action**  
Top level flowchart for this step

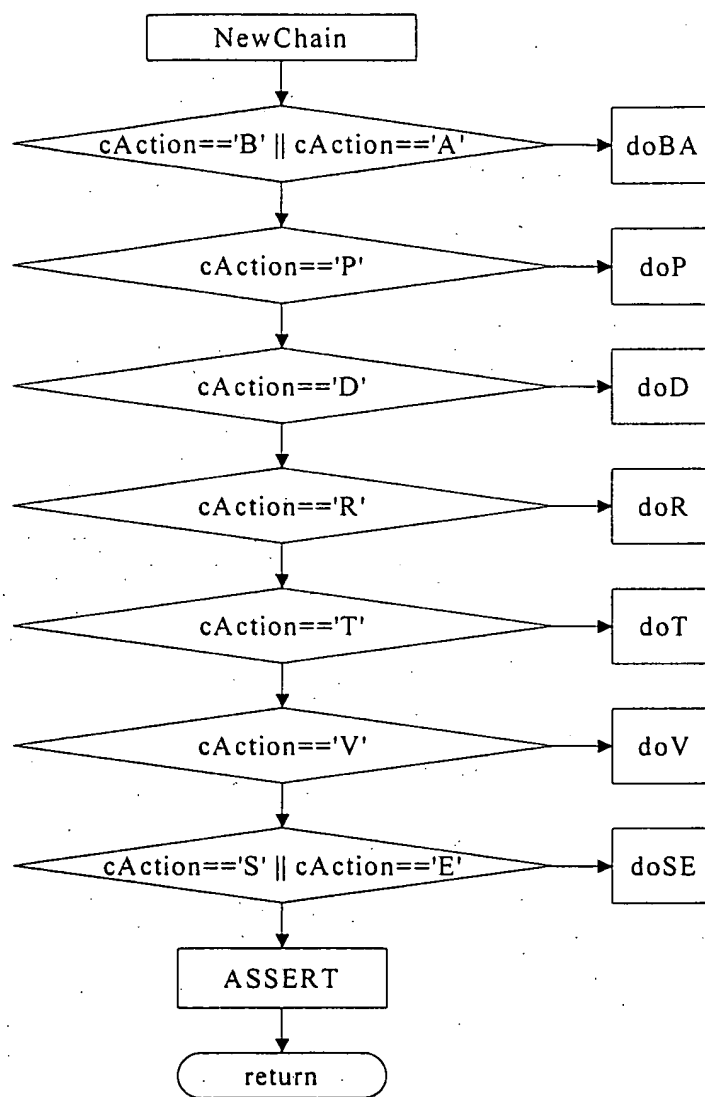


FIG. 14C

# Basic Rules for Operation in NewChain

attr: attribute  
 cA: cAction  
 SE: sourceEle  
 TE: targetEle  
 EE: emptyEle  
 AP: absolute position  
 RP: relative position  
 X(AP): X(sourceEle or targetEle or 'div') with absolute position  
 X(RP): X(sourceEle or targetEle or 'div') with relative position  
 XAP: if X(sourceEle or targetEle) has absolute position  
 XRP: if X(sourceEle or targetEle) has relative position  
 →X: insert *Element*(X) into the current chain  
 →X→Y: insert *Element*(X) and *Element*(Y) into the current chain orderly  
 AP(X): X's absolute position attribute  
 N\_C: NEW\_CHAIN(), append the new chain and set its chainBase.  
 'X': if cAction is 'X'  
 YES(X), NOT(X): X is true, false;  
 X(Y): change X's attribute Y.  
 X(AP)=X(RP)+AP(Y): add Y's absolute position attribute to X.  
 X(AP)=X(RP)+AP; add user action's AP to X  
 X(RP)=X(AP)-AP: delete X's AP attribute, then X(AP) becomes X(RP)  
 →X[→Y→Z]: insert an *Element*(X), into which two children: *Element*(Y) and *Element*(Z) are inserted.  
 →TE[cA, →SE]: when cA equals 'S', this abbreviation equals →TE[→SE→EE], when cA equals 'E', this abbreviation equals →TE[→EE→SE]  
 //X: X is comment

FIG. 14D



SE's position	TE's position	Final positon of <i>Element</i> (targetEle) or <i>Element</i> (div)
RP	RP	RP
AP	RP	RP
RP	AP	AP
AP	AP	targetEle's AP

FIG. 14E

105000-105000

### 3. Examples

After new a chain and set its chain base, insert Elements into the new chain according to the following table.

In this table, "content" means the attributes and text.

In the 2<sup>nd</sup> column of this table, "□" means the row involves absolute position.

Action	What user wants to do	What Rule Generator does
B	Insert SE(RP) before TE(RP)	$\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}$
	□ Insert SE(RP) before TE(AP)	$\text{AP(div)} = \text{AP(TE)};$ $\text{TE(RP)} = \text{TE(AP)} - \text{AP};$ $\rightarrow \text{div(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}]$
	□ Insert SE(AP) before TE(RP)	$\text{SE(RP)} = \text{SE(AP)} - \text{AP};$ $\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}$
	□ Insert SE(AP) before TE(AP)	$\text{AP(div)} = \text{AP(TE)};$ $\text{SE(RP)} = \text{SE(AP)} - \text{AP};$ $\text{TE(RP)} = \text{TE(AP)} - \text{AP};$ $\rightarrow \text{div(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{TE(RP)}]$ //use a div to wrap SE and TE
A	simliar as B	
P	□ move SE(RP) to AP	$\text{SE(AP)} = \text{SE(RP)} + \text{AP};$ $\rightarrow \text{SE(AP)};$
	□ move SE(AP) to another AP	$\text{SE(} \underline{\text{AP}} \text{)};$ $\rightarrow \text{SE(AP)};$
D	Delete this element	
R	Use SE(RP) to replace TE(RP)	$\rightarrow \text{SE(RP)};$
	□ Use SE(RP) to replace TE(AP)	$\text{SE(AP)} = \text{SE(RP)} + \text{AP(TE)};$ //SE(RP) becomes SE(AP) $\rightarrow \text{SE(AP)}$
	□ Use SE(AP) to replace TE(RP)	$\text{SE(RP)} = \text{SE(AP)} - \text{AP};$ //SE(AP) becomes SE(RP) $\rightarrow \text{SE(RP)}$
	□ Use SE(AP) to replace TE(AP)	$\text{AP(SE)} = \text{AP(TE)}$ $\rightarrow \text{SE(AP)}$
T	Change SE' attr	doD; //nothing to do with AP and RP
V	Replace SE's text	doV; //nothing to do with AP and RP
S	Insert SE(RP) to be child of TE(RP)	$\rightarrow \text{TE(RP)}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
	□ Insert SE(RP) to be child of TE(AP)	$\rightarrow \text{TE(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
	□ Insert SE(AP) to be child of TE(RP)	$\text{SE(RP)} = \text{SE(AP)} - \text{AP};$ $\rightarrow \text{TE(RP)}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
	□ Insert SE(AP) to be child of TE(AP)	$\text{SE(RP)} = \text{SE(AP)} - \text{AP};$ $\rightarrow \text{TE(AP)}[\rightarrow \text{SE(RP)} \rightarrow \text{EE}]$
E	similar as S	

FIG. 14F

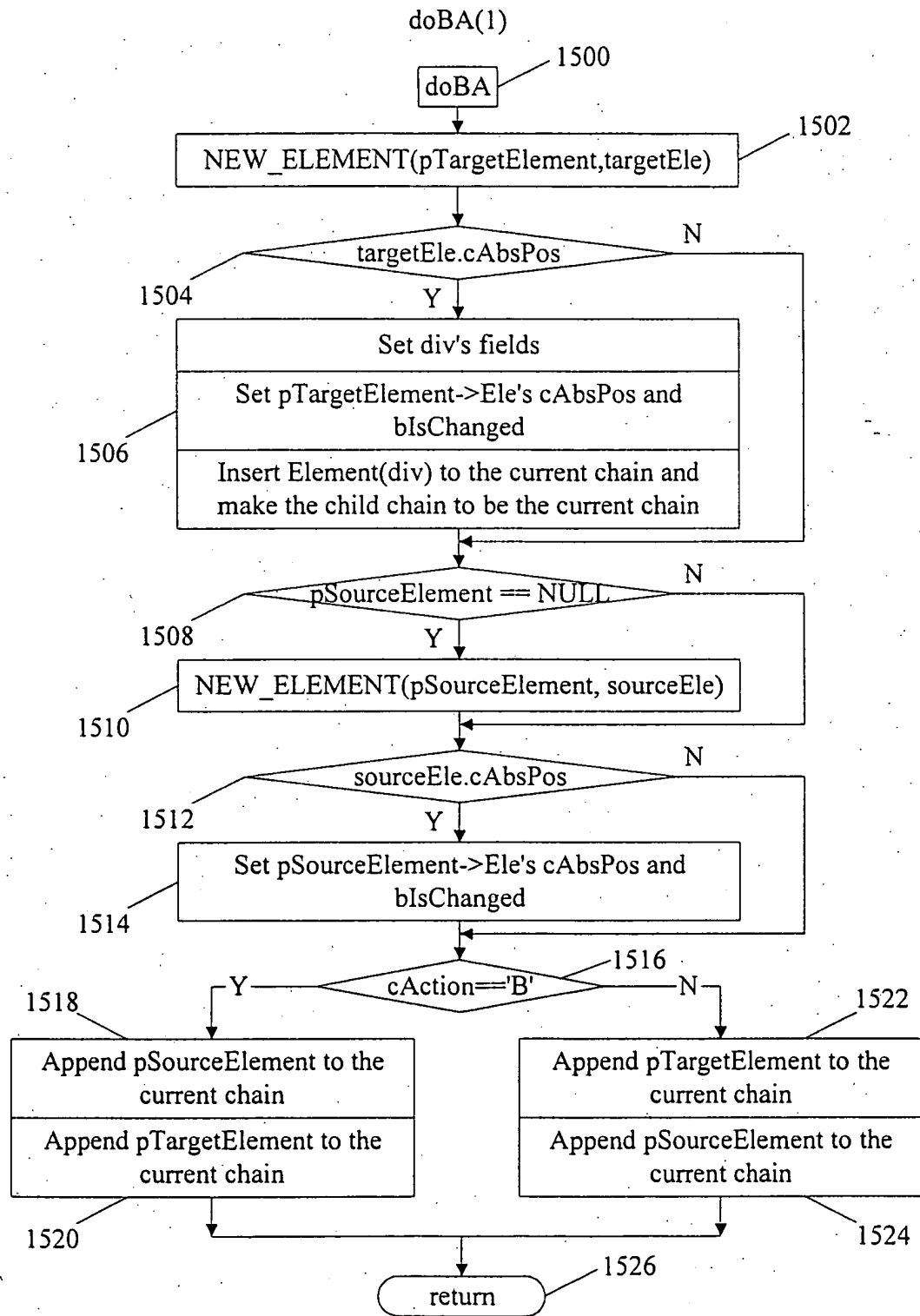


FIG. 15A

doBA(2)

Pseudo code

//if targetEle has absolute position, a div will be used to wrap sourceEle and targetEle  
//The caller of Rule Generator sets the field cAbsPos to indicate the position status of the  
element: absolute position or relative position  
NEW\_ELEMENT(pTargetElement, targetEle);

```
if(targetEle.cAbsPos){
    //set tag name
    div.sNewTag = "div";
    //Because targetEle does not exist in any chain, the reference absolute position is used for
    div
    div.cAbsPos = REF_ABS_POS;
    div.sAbsPos = targetEle.sPath;
    //Set div's sFrame and sPath
    div.sFrame = targetEle.sFrame;
    div.sPath = targetEle.sPath;
    //set bIsAbsPosOrg and cAbsPos to indicate that the absolute position attribute shall not be
    output when xslt is applied.
    pTargetElement->Ele.cAbsPos = NO_ABS_POS;

    //set bIsChanged to indicate targetEle is changed.
    pTargetElement->Ele.bIsChanged = true;

    //new an Element for div and append to the current new chain
    NEW_ELEMENT(pDivElement, targetEle);
    pDivElement->bIsChainBase= true;
    APPEND_ELEMENT(pChain, pDivElement);

    //make a new chain to be the child chain of div
    pChain = new chain;
    pDivElement->pChildChain = pChain;
}
else !!!
    pTargetElement->bIsChainBase = true;

//if sourceEle does not exist in any chain, make a new Element for sourceEle
if( ! pSourceElement){
    NEW_ELEMENT(pSourceElement, sourceEle);
}
```

FIG. 15B

```

doBA(3)
//if sourceEle is absolute position, set blsAbsPosOrg, cAbsPos and blsChanged to indicate that
the absolute position attribute of sourceEle shall not be output and sourceEle is changed.
if(sourceEle.cAbsPos){
    pSourceElement->Ele.cAbsPos = NO_ABS_POS;
    pSourceElement->Ele.blsChanged = true;
}

//append pSourceElement and pTargetElement according to the action
if(cAction == 'B'){
    APPEND_ELEMENT(pChain, pSourceElement);
    APPEND_ELEMENT(pChain, pTargetElement);
}
else{
    APPEND_ELEMENT(pChain, pTargetElement);
    APPEND_ELEMENT(pChain, pSourceElement);
}

```

FIG. 15C

```
pSourceElement->bIsChainBase = true;
APPEND_ELEMENT(pChain, pSourceElement);
```

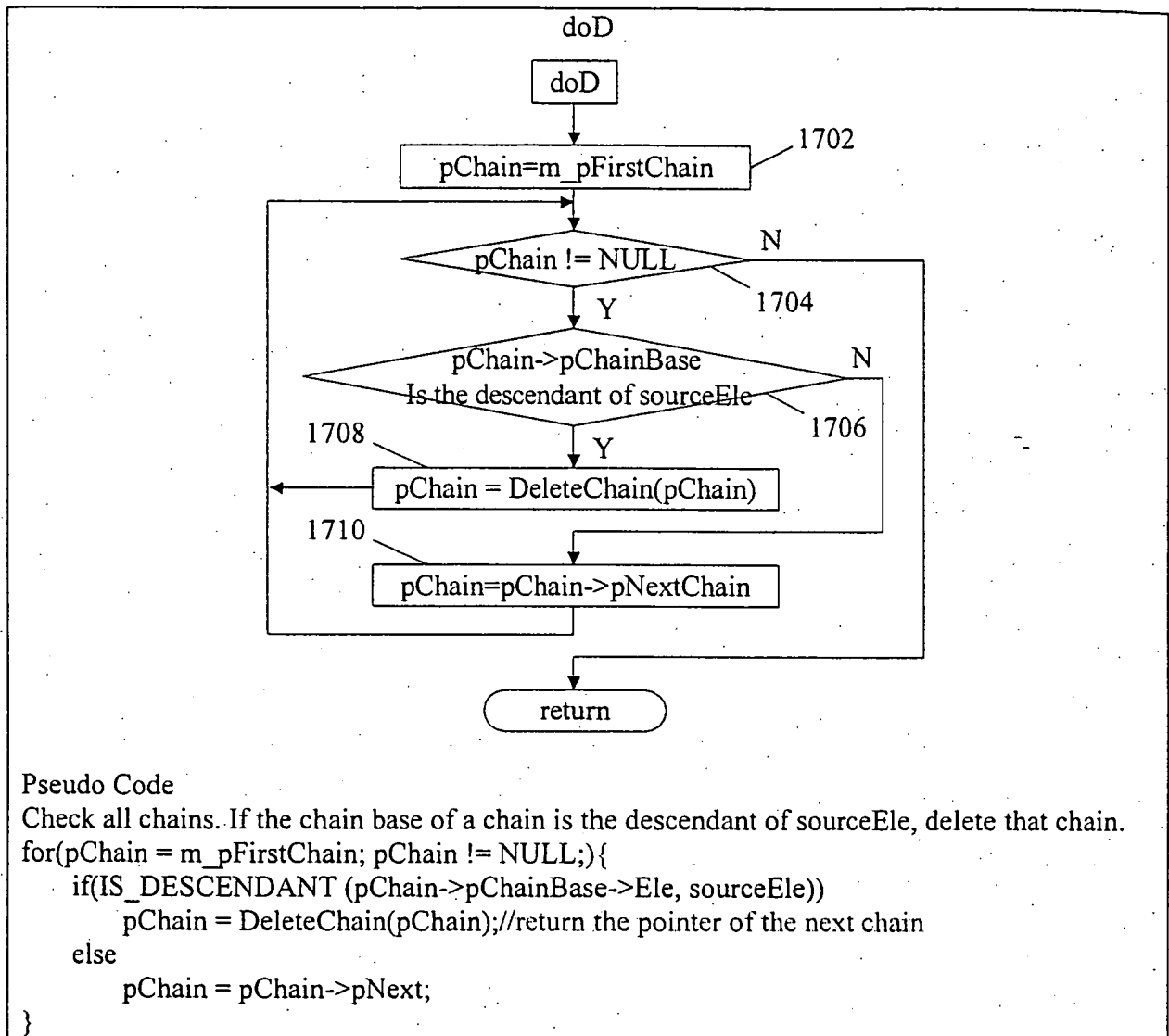
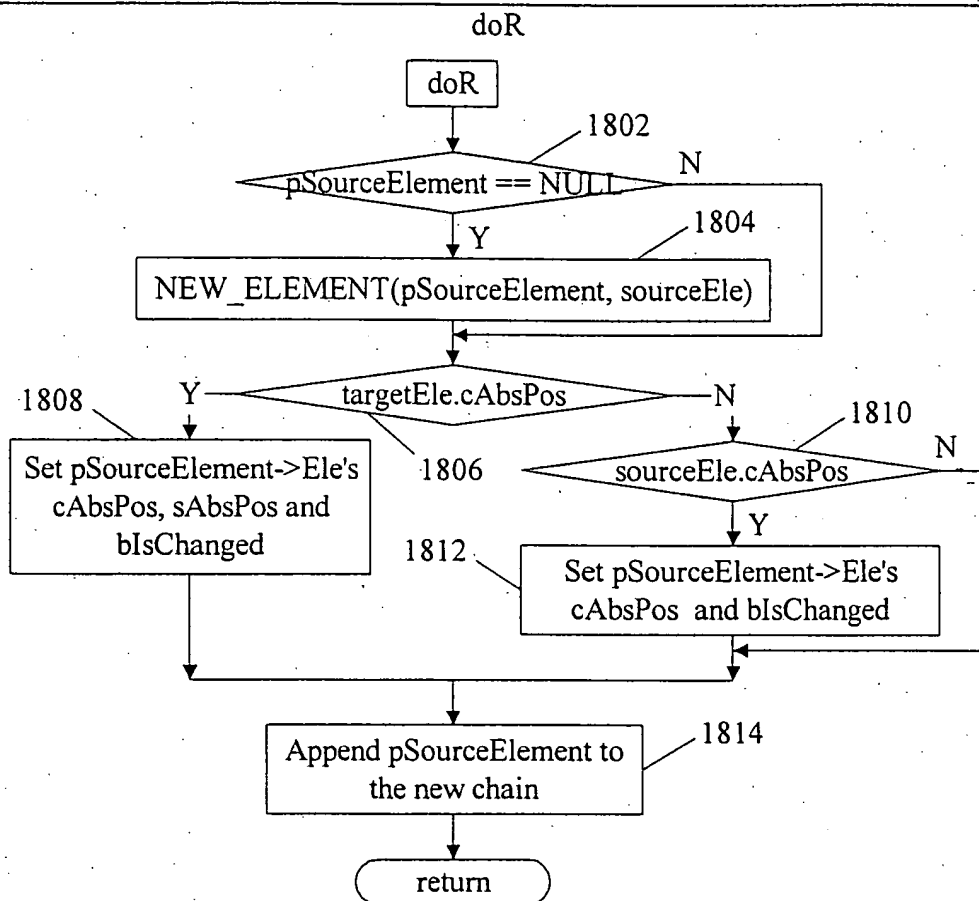


FIG. 17



Pseudo code

```

if( ! pSourceElement ){
    NEW_ELEMENT(pSourceElement, sourceEle);
}
if(targetEle.cAbsPos){

    pSourceElement->Ele.cAbsPos = REF_ABS_POS;
    pSourceElement->Ele.sAbsPos = targetEle.sPath;
    pSourceElement->Ele.blsChanged = true;
}
else{
    if(sourceEle.cAbsPos){
        pSourceElement->Ele.cAbsPos = NO_ABS_POS;
        pSourceElement->Ele.blsChanged = true;
    }
}

APPEND_ELEMENT(pChain, pSourceElement);
  
```

FIG. 18



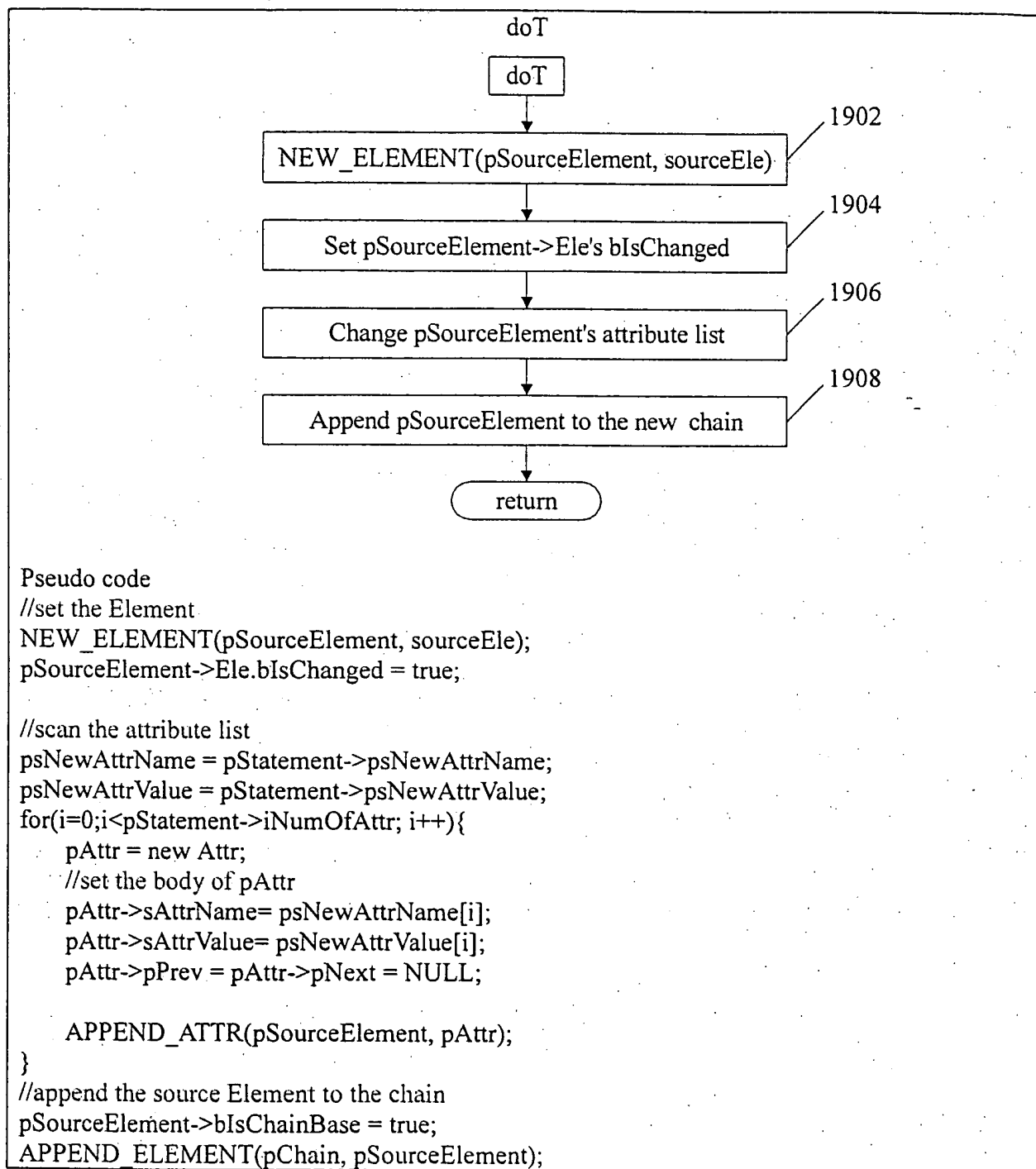


FIG. 19

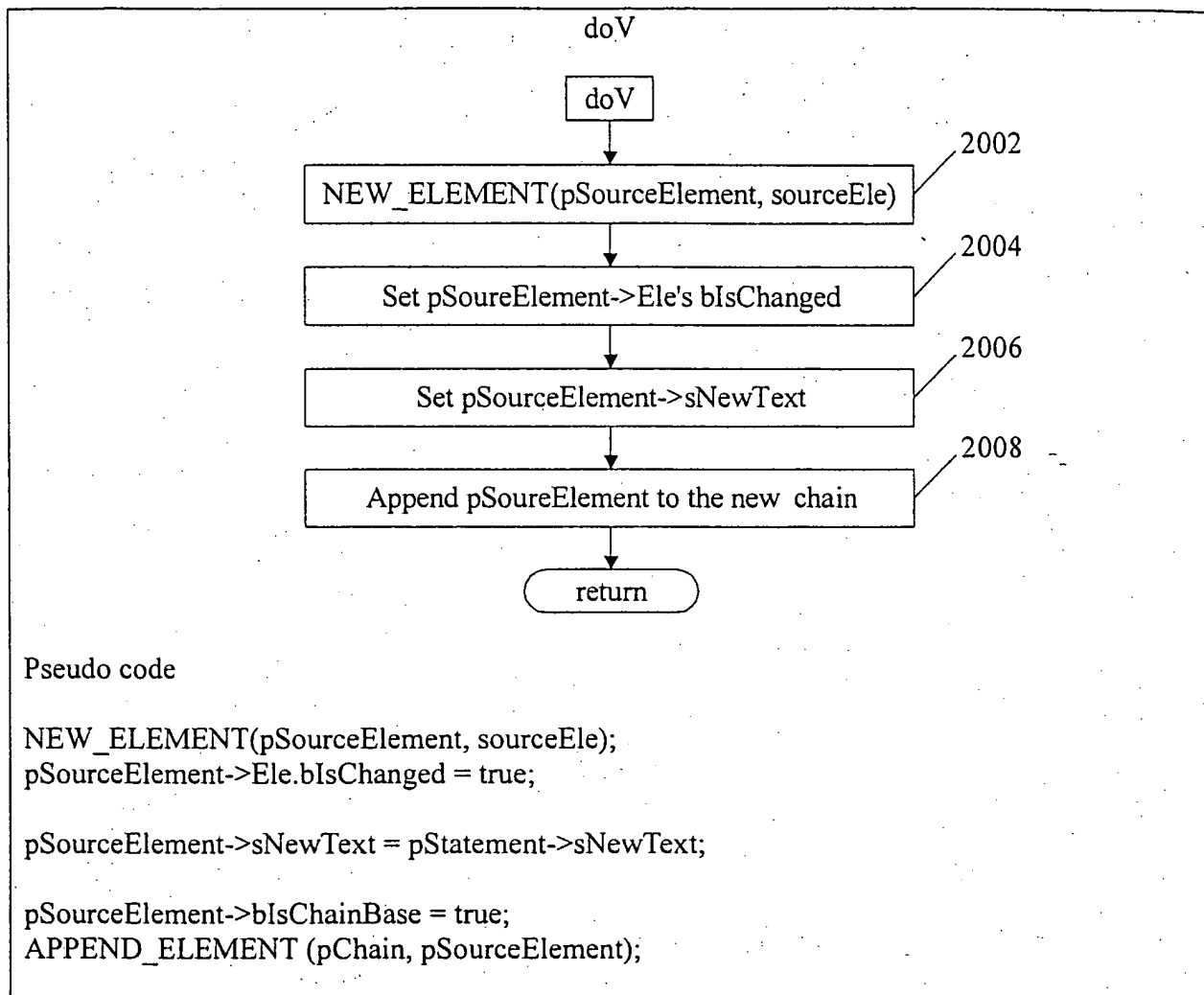


FIG. 20

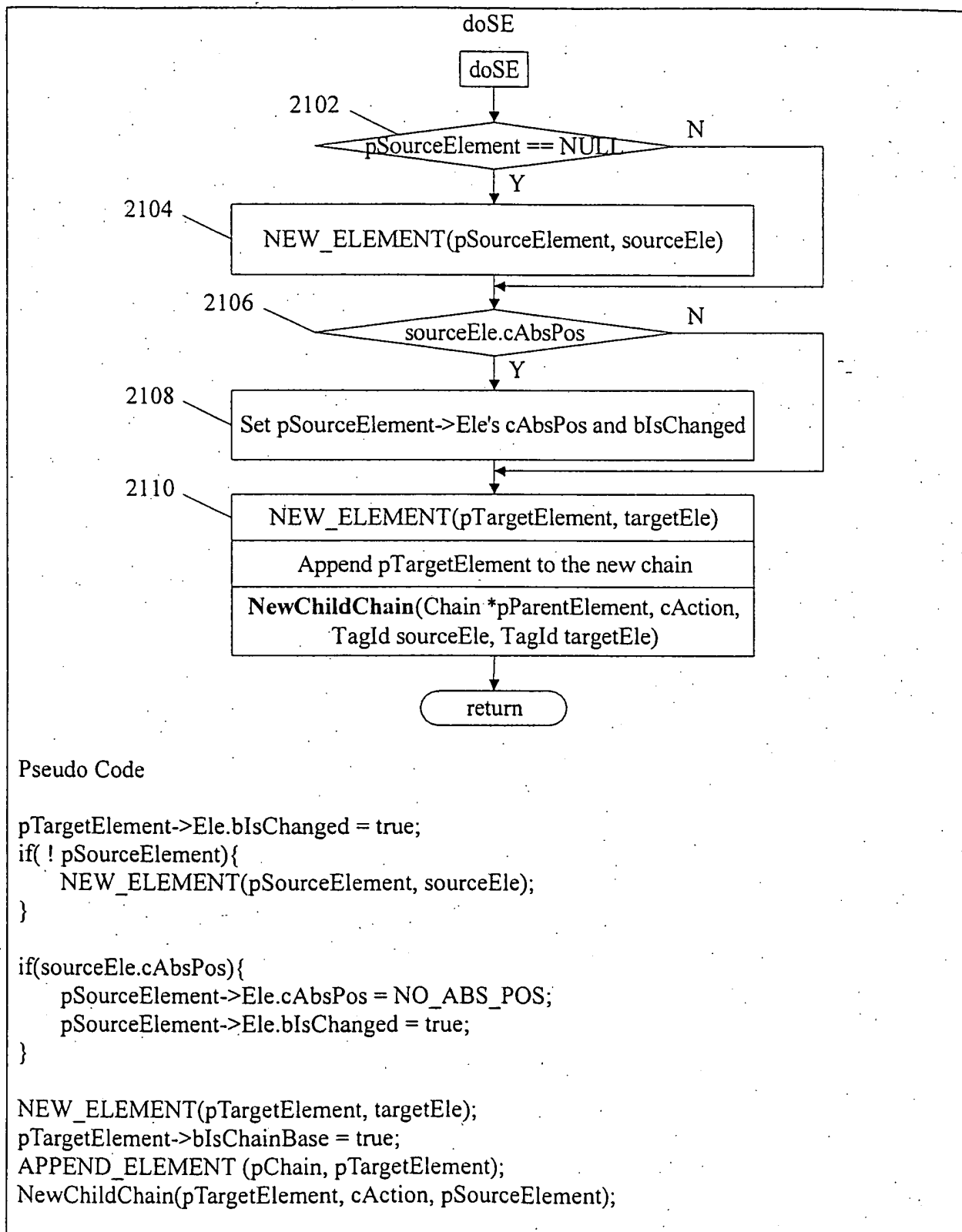


FIG. 21

### NewChildChain

Pseudo code

```
//New a child chain for the input Element
NEW_CHILD_CHAIN(pChain);
pParentElement->pChildChain = pChain;
pChain->pParentElement = pParentElement;
//New a empty Element
emptyEle.iFamilyId = -1; //-1 represent this is a empty Element
NEW_ELEMENT(pEmptyElement, emptyEle);

//Append two Elements according to the action
if(cAction == 'S'){
    APPEND_ELEMENT(pChain, pSourceElement);
    APPEND_ELEMENT(pChain, pEmptyElement);
}
else{//cActio == 'E'
    APPEND_ELEMENT(pChain, pEmptyElement);
    APPEND_ELEMENT(pChain, pSourceElement);
}
```

FIG. 22

UpdateChain

Set local variables 2302

Cut the source element from original place, if it exists 2304

Update deleted chain 2306

Update the chain according to user action 2308

doBA 2310

doP 2312

doD 2314

doR 2316

doT 2318

doV 2320

doSE 2322

FIG. 23A

### UpdateChain(1)

#### Step 1 Set local variables

```
sourceEle = pStatement->sourceEle; targetEle = pStatement->targetEle;  
cAction = pStatement->cAction;  
pSourceElement = pElementInfo->pSourceElement;  
pTargetElement = pElementInfo->pTargetElement;
```

```
pChainForSourceEle = pElementInfo->pChainForSourceEle;  
pChainForTargetEle = pElementInfo->pChainForTargetEle;
```

#### Step 2 Cut the source Element from the original place if it exists(the same as that in NewChain)

```
if(pSourceElement &&  
    (cAction == 'B' || cAction == 'A' || cAction == 'S' || cAction == 'E' ||  
     cAction == 'R' || cAction == 'D')  
){  
    //Cut the Element from the original place.  
    CUT(pElementInfo->pChainForSourceEle->pFirstElement,  
        pElementInfo->pChainForSourceEle->pLastElement,  
        pSourceElement);  
}
```

//other actions are PTV

#### Step 3 Update Deleted Chain (the same as that in NewChain)

```
switch(cAction){  
    case 'R': UpdateDelChain(sourceEle, cAction);  
              UpdateDelChain(targetEle, cAction);break;  
    case 'T':  
    case 'V': break;  
    default: UpdateDelChain(sourceEle, cAction); break;// 'B' 'A' 'S', 'E', 'D'  
}
```

FIG. 23B

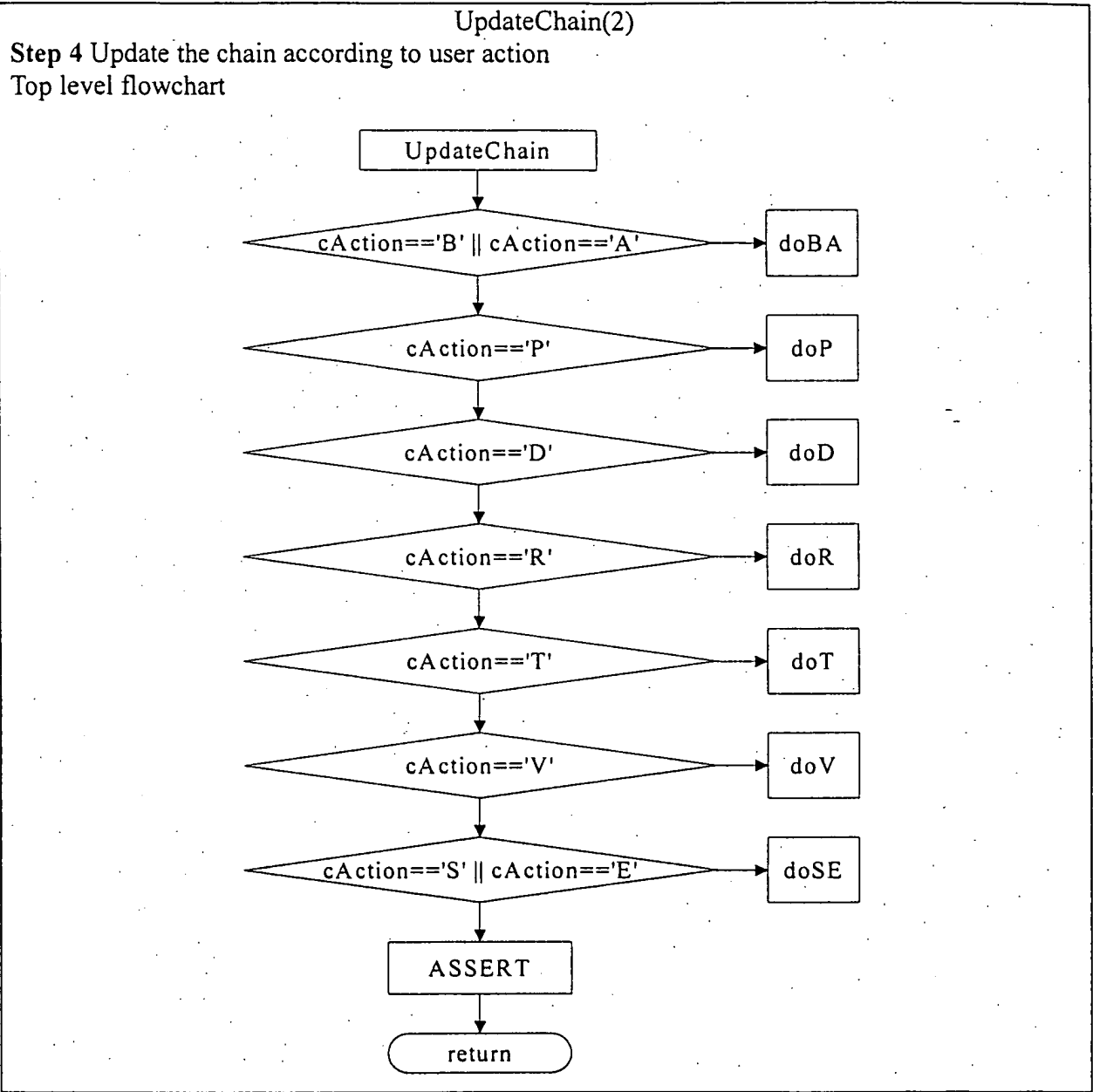


FIG. 23C

### Basic rules for the above operation in UpdateChain

#### 1. Abbreviation

Two abbreviations are defined besides the ones in NewChain

$\rightarrow\{X\}$ : X is an Element that is already in the current chain.

$\rightarrow X \rightarrow \forall$ : Replace *Element*(Y) with *Element*(X).

#### 2. Handle different positions(the same as that in NewChain)

#### 3. Examples

Action	What user wants to do	What Rule Generator does
B	Insert SE(RP) before TE(RP)	$\rightarrow SE(RP) \rightarrow \{TE(RP)\}$
	<input type="checkbox"/> Insert SE(RP) before TE(AP)	AP(div)=AP(TE); TE(RP)=TE(AP)-AP; $\rightarrow div(AP)[\rightarrow SE(RP) \rightarrow TE(RP)] \rightarrow TE(AP)$
	<input type="checkbox"/> Insert SE(AP) before TE(RP)	SE(RP)=SE(AP)-AP; //then SE(AP) becomes SE(RP) $\rightarrow SE(RP) \rightarrow \{TE(RP)\}$
	<input type="checkbox"/> Insert SE(AP) before TE(AP)	AP(div)=AP(TE); SE(RP)=SE(AP)-AP; TE(RP)=TE(AP)-AP; $\rightarrow div(AP)[\rightarrow SE(RP) \rightarrow TE(RP)] \rightarrow TE(AP)$ //use a div to wrap SE and TE
A	similiar as B	
P	<input type="checkbox"/> move SE(RP) to AP	SE(AP)=SE(RP)+AP;
	<input type="checkbox"/> move SE(AP) to another AP	SE(AP);
D	Delete this element	
R	Use SE(RP) to replace TE(RP)	$\rightarrow SE(RP) \rightarrow TE(RP)$ ;
	<input type="checkbox"/> Use SE(RP) to replace TE(AP)	SE(AP)=SE(RP)+AP(TE); //SE(RP) becomes SE(AP) $\rightarrow SE(AP) \rightarrow TE(AP)$
	<input type="checkbox"/> Use SE(AP) to replace TE(RP)	SE(RP)=SE(AP)-AP; //SE(AP) becomes SE(RP) $\rightarrow SE(RP) \rightarrow TE(RP)$
	<input type="checkbox"/> Use SE(AP) to replace TE(AP)	AP(SE)=AP(TE) $\rightarrow SE(AP) \rightarrow TE(AP)$
T	Change SE' attr	doD; //nothing to do with AP and RP
V	Replace SE's text	doV; //nothing to do with AP and RP
S	Insert SE(RP) to be child of TE(RP)	$\rightarrow \{TE(RP)\}[\rightarrow SE(RP) \rightarrow EE]$
	<input type="checkbox"/> Insert SE(RP) to be child of TE(AP)	$\rightarrow \{TE(AP)\}[\rightarrow SE(RP) \rightarrow EE]$
	<input type="checkbox"/> Insert SE(AP) to be child of TE(RP)	SE(RP)=SE(AP)-AP; $\rightarrow \{TE(RP)\}[\rightarrow SE(RP) \rightarrow EE]$
	<input type="checkbox"/> Insert SE(AP) to be child of TE(AP)	SE(RP)=SE(AP)-AP; $\rightarrow \{TE(AP)\}[\rightarrow SE(RP) \rightarrow EE]$
E	similar as S	

FIG. 23D



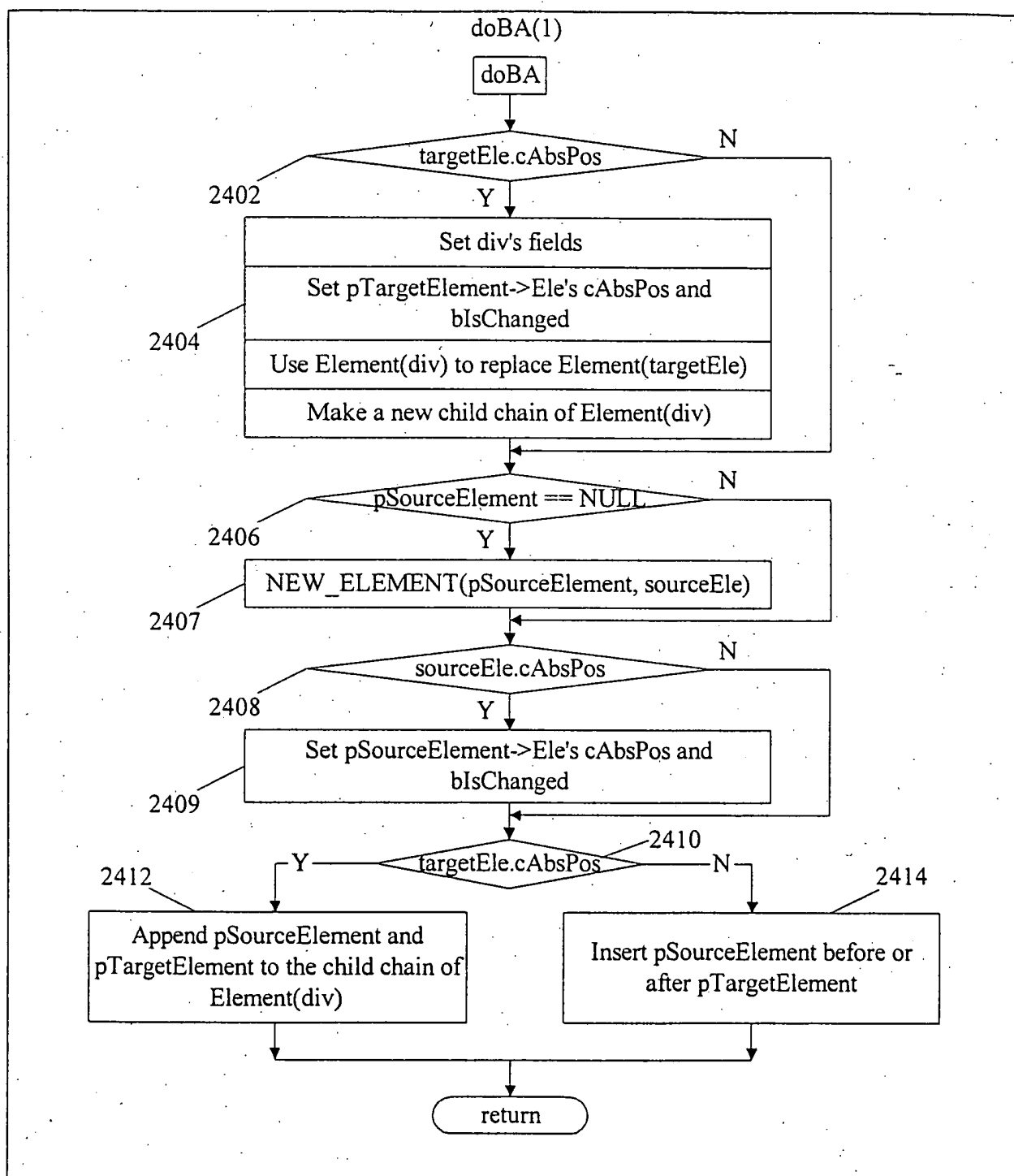


FIG. 24A-1

doBA(1)

Pseudo code

```
//if targetEle has absolute position, a div will be used to wrap sourceEle and targetEle
//The caller of Rule Generator sets the field cAbsPos to indicate the position status of the
element: absolute position or relative position
if(targetEle.cAbsPos){
    //set tag name
    div.sNewTag = "div";
    //set targetEle's absolute position attribute to div
    div.cAbsPos = targetEle.cAbsPos;
    //when cAbsPos is REF_ABS_POS, sPath is used, otherwise (x, y) is used.
    div.sAbsPos = targetEle.sAbsPos
```

FIG. 24A-2

```

doBA(2)

div.x = targetEle.x;
div.y = targetEle.y;
//set targetEle.cAbsPos to NO_ABS_POS indicate that the absolute position attribute shall
not be output when xslt is applied.
pTargetElement->Ele.cAbsPos = NO_ABS_POS;
//set blsChanged to indicate targetEle is changed.
pTargetElement->Ele.blsChanged = true;
//new an Element for div and append to the current new chain
NEW_ELEMENT(pDivElement, div);
//use Element(div) to replace Element(targetEle)
REPLACE(pChainForTargetEle->pFirstElement, pChainForTargetEle->pLastElement,
pTargetElement, pDivElement);
//make a new chain to be the child chain of div
pDivChain = new chain;
pDivElement->pChildChain = pDivChain;
}
//if sourceEle does not exist in any chain, make a new Element for sourceEle
if(! pSourceElement){ NEW_ELEMENT(pSourceElement, sourceEle); }
//if sourceEle is absolute position, set blsAbsPosOrg, cAbsPos and blsChanged to indicate that
the absolute position attribute of sourceEle shall not be output and sourceEle is changed.
if(sourceEle.cAbsPos){
    pSourceElement->Ele.cAbsPos = NO_ABS_POS;
    pSourceElement->Ele.blsChanged = true;
}
if(targetEle.cAbsPos){
    if(cAction == 'B'){
        APPEND_ELEMENT(pDivChain, pSourceElement);
        APPEND_ELEMENT(pDivChain, pTargetElement);
    }
    else{
        APPEND_ELEMENT(pDivChain, pTargetElement);
        APPEND_ELEMENT(pDivChain, pSourceElement);
    }
}
else{
    if(cAction == 'B'){
        INSERT_BEFORE(pChainForTargetEle->pFirstElement, pSourceElement, pTargetElement);
    }
    else{
        INSERT_AFTER(pChainForTargetEle->pFirstElement, pTargetElement, pSourceElement);
    }
}
}

```

FIG. 24B

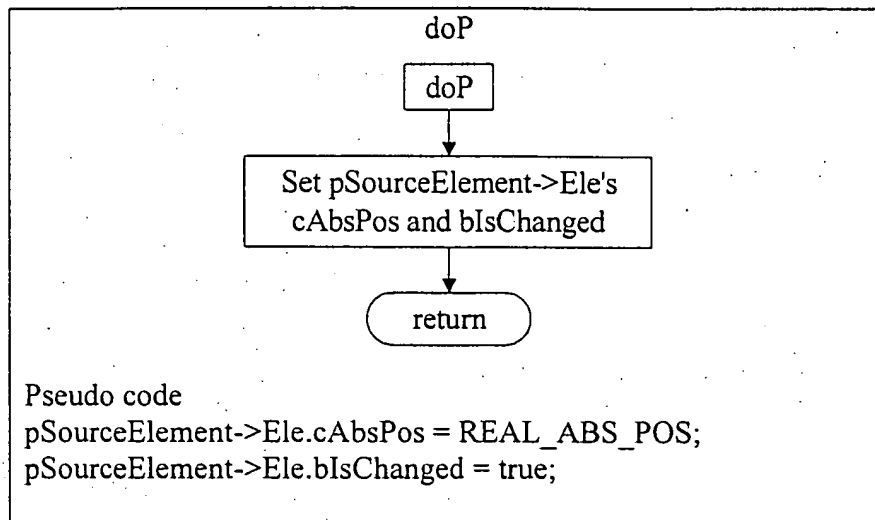
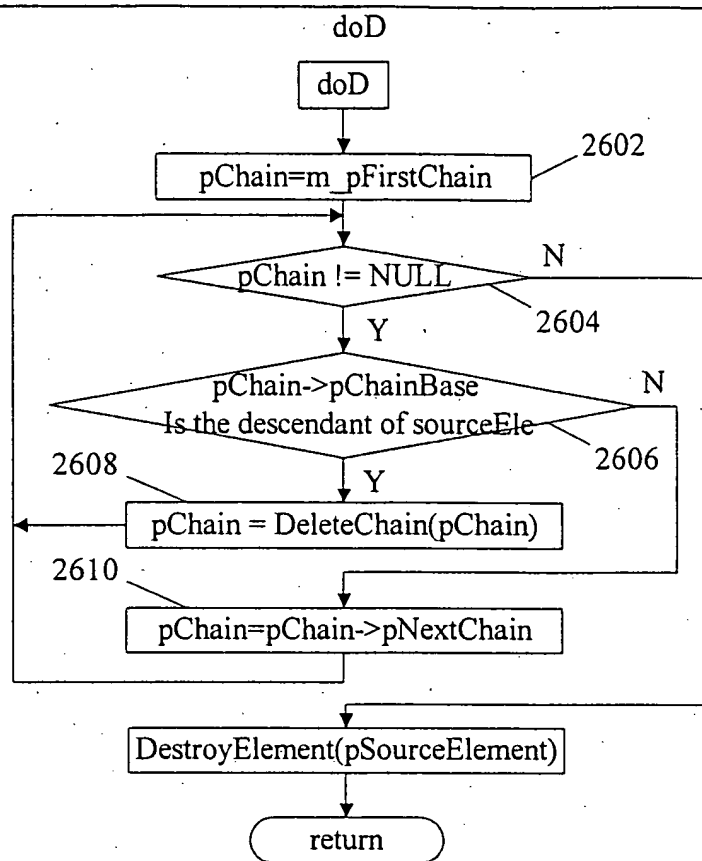


FIG. 25



Pseudo Code

```

for(pChain = m_pFirstChain; pChain != NULL;){
    if(IS_DESCENDANT(pChain->pChainBase->Ele, pSourceElement->Ele))
        pChain = DeleteChain(pChain);//return the pointer of the next chain
    else
        pChain = pChain->pNext;
}
//pSourceElement has been cut from the original place and it is totally destroyed now.
DestroyElement(pSourceElement);
  
```

FIG. 26

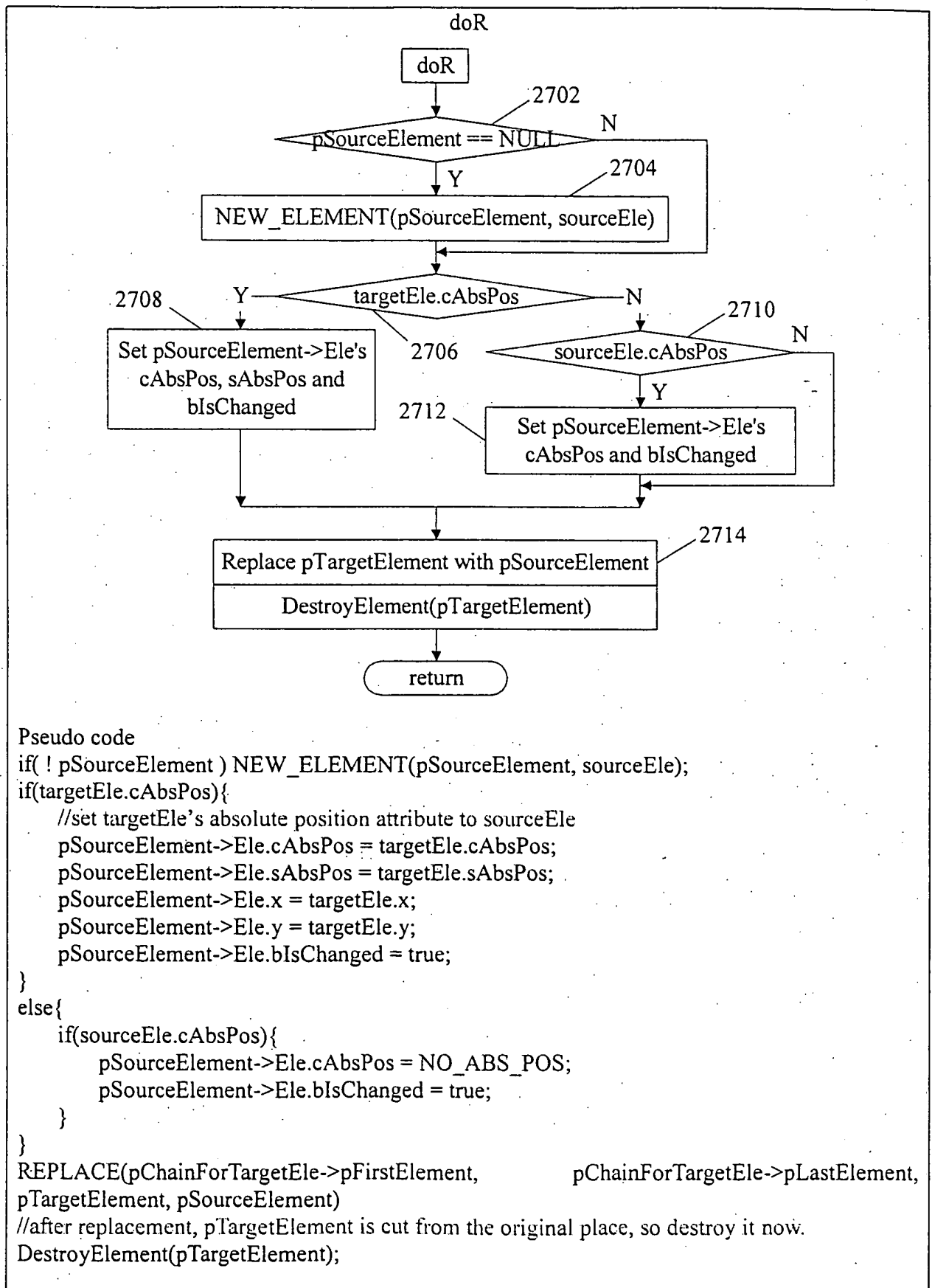


FIG. 27

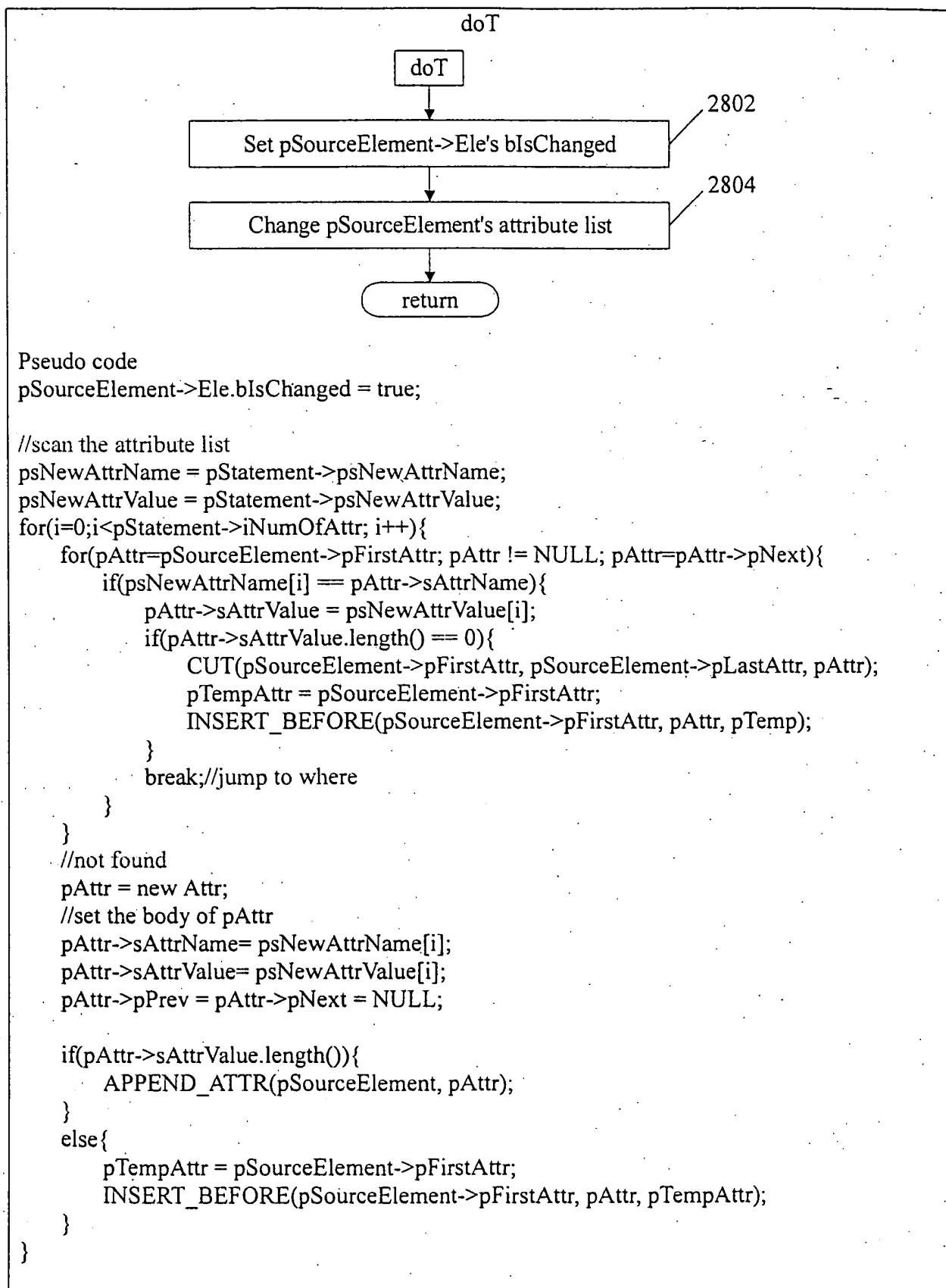


FIG. 28

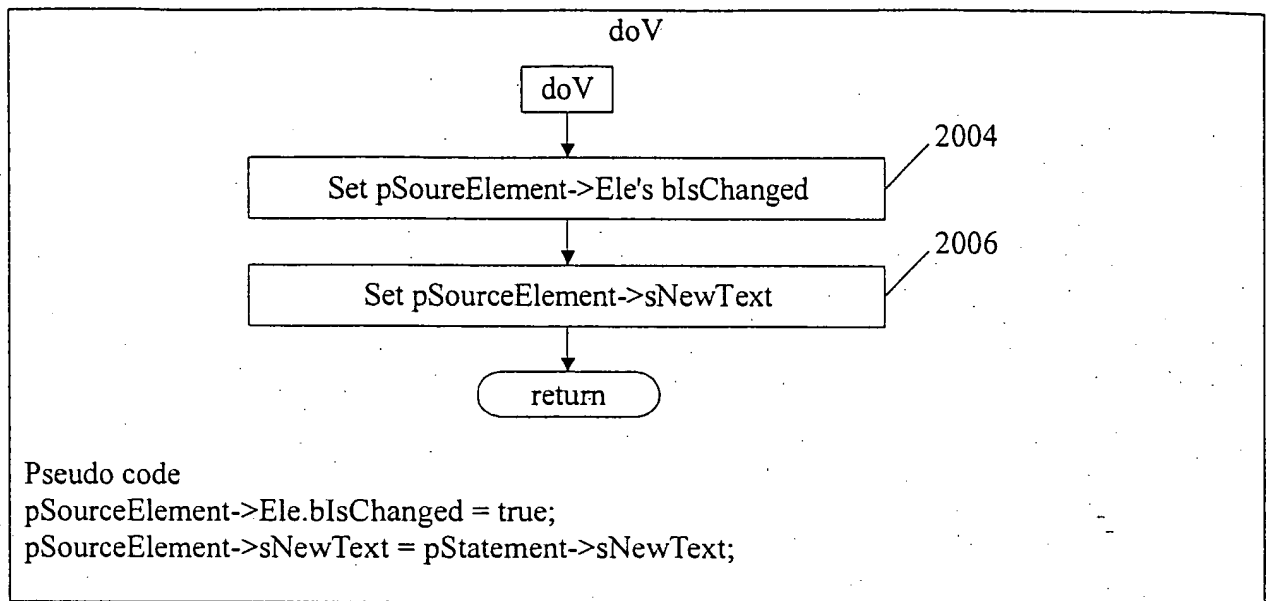
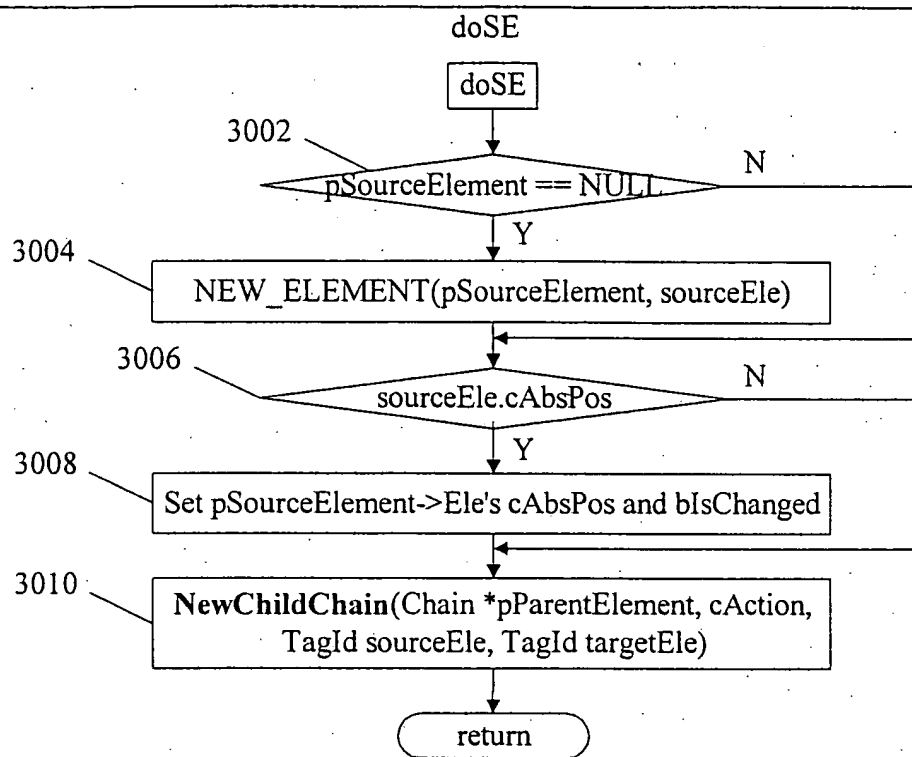


FIG. 29





Pseudo code

pTargetElement->Ele.bIsChanged = true;

if( ! pSourceElement) NEW\_ELEMENT(pSourceElement, sourceEle);

if(sourceEle.cAbsPos){

    pSourceElement->Ele.cAbsPos = NO\_ABS\_POS;

    pSourceElement->Ele.bIsChanged = true;

}

if(pTargetElement->pChildChain){

    if(cAction == 'S'){

        pElement = pTargetElement->pChildChain->pFirstElement;

        INSERT\_BEFORE(pTargetElement->pChildChain->pFirstElement, pSourceElement, pElement);

    }

    else{

        APPEND\_ELEMENT(pTargetElement->pChildChain, pSourceElement);

    }

}

else

    NewChildChain(pTargetElement, cAction, pSourceElement);

FIG. 30

```

UpdateDelChain
pRefElement = m_pDelChain->pFirstElement;
while(pRefElement != NULL){
    if(IS_EQUAL(pRefElement->Ele, Ele)){
        pRefElement->cAction = cAction;
        return;
    }
    pRefElement = pRefElement ->pNext;
}

NEW_ELEMENT(pElement, Ele);
APPEND_ELEMENT(m_pDelChain, pElement);
return;

```

FIG. 31

```

FilterDelChain
pCurrElement = m_DelChain->pFirstElement;
while(pCurrElement != NULL){
    if(Element is from source page){
        //look for its yougest ancestor
        bHasAncestor = false;
        pRefElement = m_pDelChain->pFirstElement;
        for( ; pRefElement != NULL; pRefElement = pRefElement->pNext) {
            if(pRefElement != pCurrElement){
                if (IS_DESCENDANT(pRefElement->Ele, pCurrElement->Ele)){
                    if(bHasAncestor){
                        if(IS_DESCENDANT(pYougestAncestor->Ele, pRefElement->Ele)){
                            pYougestAncestor = pRefElement;
                        }
                    }
                    else{
                        bHasAncestor = true;
                        pYougestAncestor = pRefElement;
                    }
                }
            }
        }
        if(bHasAncestor )
            if(pYougestAncestor->cAction != 'D')
                //Note: when put an exist Element into Deleted Chain, that action of Element is overwritten.
                pCurrElement->Ele.cAbsPos = false; //mark it not to be deleted
            else
                pCurrElement->Ele.cAbsPos = true;
        else
            pCurrElement->Ele.cAbsPos = true; //has no ancestor, mark it to be deleted later
        } //if(element is in source page
        else
            pCurrElement->Ele.cAbsPos = false;
        pCurrElement = pCurrElement->pNext;
    }
}

//Remove Elements whose cAbsPos = true
pCurrElement=m_pDelChain->pFirstElement;
while(pCurrElement != NULL){
    if(pCurrElement->Ele.cAbsPos)
        pCurrElement=DeleteElement(&m_DelChain, pCurrElement);
    else
        pCurrElement=pCurrElement->pNext;
}

```

FIG. 32

### AssemblyChain

```

//Assembly chains cards and pages according to the field iPage and iCard of structure TagId
for( pChain = m_pFirstChain; pChain != NULL; pChain = pChain->pNext){
    for(pPage = m_pFirstPage; pPage != NULL; pPage = pPage->pNext){
        if(pChain->pChainBase->Ele.iPage == pPage->iPage){
            for( pCard = pPage->pFirstCard; pCard != NULL; pCard = pCard->pNext){
                if(pChain->pChainBase->Ele.iCard == pCard->iCard){
                    CUT(m_pFirstChain, m_pLastChain, pChain);
                    APPEND_CHAIN_TO_CARD(pCard, pChain);

                    if(pCard->iEntry == HAS_NOT_ENTRY)
                        pCard->iEntry = pChain->pChainBase->Ele.iFamilyId;
                    goto NEXT_CHAIN;
                }
            }
        }
    }
    NEW_PAGE(pPage, pChain->pChainBase->Ele.iPage)
    APPEND_PAGE(this, pPage);
    NEW_CARD(pCard, pChain->pChainBase->Ele.iCard);
    APPEND_CARD(pPage, pCard);
    CUT(m_pFirstChain, m_pLastChain, pChain);
    APPEND_CHAIN_TO_CARD(pCard, pChain);

    if(pCard->iEntry == HAS_NOT_ENTRY){
        pCard->iEntry = pChain->pChainBase->Ele.iFamilyId;
    }
NEXT_CHAIN: ;
}

//Assembly Elements in Deleted Chains according to FamilyId.
for(pElement = m_pDelChain->pFirstElement; pElement != NULL; pElement = pElement-
->pNext){
    for(pChain = m_pFirstDelChain; pChain != NULL; pChain = pChain->pNext){
        if(pElement->Ele.iFamilyId == pChain->pFirstElement->Ele.iFamilyId){
            CUT(m_pDelChain->pFirstElement, m_pDelChain->pLastElement, pElement);
            APPEND_ELEMENT(pChain, pElement);
            goto NEXT_DEL_ELEMENT;
        }
    }
    NEW_CHILD_CHAIN(pChain);
    APPEND_DEL_CHAIN(pChain);
    CUT_ELEMENT(m_pDelChain->pFirstElement, m_pDelChain->pLastElement, Element);
    APPEND_ELEMENT(pChain, pElement);
NEXT_DEL_ELEMENT:;
}

```

FIG. 33

# ParseFrame(1)

```

if(pChain->pChainBase)
    pElement = pChain->pChainBase;
else
    pElement = pChain->pFirstElement;

bTmpHasFrame = bSrcHasFrame = true;

while(pElement != NULL && (bSrcHasFrame || bTmpHasFrame)){

    if(pElement->bIsChainBase) bChainBaseIsFound = true;

    if(pElement->Ele.sPath.charAt(0) == 't'){
        cFrom = TEMPLATE;
        if(bTmpHasFrame)
            pVar = pPage->pRootTmpVar;
        else
            goto NEXT_ELEMENT;
    }
    else if(pElement->Ele.sPath.charAt(0) == 's'){
        cFrom = SOURCE;
        if(bSrcHasFrame)
            pVar = pPage->pRootSrcVar;
        else
            goto NEXT_ELEMENT;
    }
    else{
        //DEBUG
        printf("wrong path!\n");
    }

    sFrame = pElement->Ele.sFrame;
    iMaxLayer = GetLayerNumber(sFrame);
}

```

FIG. 34A

ParseFrame(2)

```

for(i=0;i<iMaxLayer;i++){

    iFirstDotPos = GetFirstCharPos(sFrame, '.');

    if(iFirstDotPos != -1){
        sTemp = sFrame.substringData(0, iFirstDotPos);
        sFrame = sFrame.substringData(iFirstDotPos+1, sFrame.length()-iFirstDotPos);
    }
    else
        sTemp = sFrame;

    if(sTemp.charAt(0) == 'i'){
        bIsFrame = false;
        sTemp = sTemp.substringData(1, sTemp.length()-1);
    }
    else
        bIsFrame = true;

    if(sTemp.length() == 0){//the page containing this element has no frame.
        if(cFrom == TEMPLATE)
            bTmpHasFrame = false;
        else
            bSrcHasFrame = false;

        goto NEXT_ELEMENT;
    }
    else
        iFrameIndex = atoi(sTemp.transcode());

    if(bIsFrame){
        if(iFrameIndex < pVar->iMaxFrame){
            pVar = pVar->pFirstFrame;
            for(i=1;i<iFrameIndex; i++){
                pVar = pVar->pNext;
            }
            goto NEXT_ELEMENT;
        }
        else{
            for(j = pVar->iMaxFrame; j < iFrameIndex; j++){
                itoa(j+1, szBuffer, 10);
                sCurrFrame = szBuffer;
            }
        }
    }
}

```

FIG. 34B

```

                                ParseFrame(3)
                                if(pVar != pPage->pRootSrcVar && pVar != pPage->pRootTmpVar)
                                    sCurrFrame = pVar->sFrame + "." + sCurrFrame;

                                NEW_VAR(pNewVar, ScurrFrame);
                                APPEND(pVar, pNewVar, Frame);
                                }//loop for j
                                pVar->iMaxFrame = iFrameIndex;
                                pVar = pVar->pLastFrame;
                                }
                                }
                                else{
                                    if(iFrameIndex < pVar->iMaxIFrame){
                                        pVar = pVar->pFirstIFrame;
                                        for(i=1; i<iFrameIndex; i++){
                                            pVar = pVar->pNext;
                                        }
                                        goto NEXT_ELEMENT;
                                    }
                                    else{
                                        for(j = pVar->iMaxIFrame; j < iFrameIndex; j++){
                                            itoa(j+1, szBuffer, 10);
                                            sCurrIFrame = szBuffer;

                                            if(pVar != pPage->pRootSrcVar && pVar != pPage->pRootTmpVar)
                                                sCurrFrame = pVar->sFrame + "." + sCurrFrame;

                                            NEW_VAR(pNewVar, sCurrIFrame);
                                            APPEND(pVar, pNewVar, IFrame);
                                        }
                                        pVar->iMaxIFrame = iFrameIndex;
                                        pVar = pVar->pFirstIFrame;
                                    }
                                }
                                pVar->bToOutput = true;
                                }
                                NEXT_ELEMENT:

                                if(pChain->pChainBase == pElement)
                                    pElement = pChain->pFirstElement;
                                else
                                    pElement = pElement->pNext;
                                }//end of loop for pElement

```

FIG. 34C

OutputVar

```
if(pVar->bToOutput == false) return;

if(pVar->sFrame contains the character '.'){
    iLastDotPos = pVar->sFrame.lastIndexOf(pVar->sFrame, '.');
    iLength = pVar->sFrame.length();

    sParentFrame = pVar->sFrame.substringData(0, iLastDotPos);
    sSelfFrame = pVar->sFrame.substringData(iLastDotPos + 1, iLength - iLastDotPos-1);
}
else{
    sParentFrame = "";
    sSelfFrame = pVar->sFrame;
}

if(the first character of sSelfFrame is 'i'){
    //it is a "iframe"
    bIsIFrame = true;
    sSelfFrame = pVar->sFrame.substringData(iLastDotPos + 1, iLength - iLastDotPos - 1);
}

//Set frame type
sFrameType = bIsFrame ? "iframe" : "frame"

//Output the variable for this frame
<<xsl:variable name="sFrom" pVar->sFrame" select="document("$"
    sFrom sParentFrame//sFrameType [ sSelfFrame ]/@src)" />

//Iteratively output the XSLT variables for the frames of the current var
for(pFrame = pVar->pFirstFrame; pFrame ; pFrame = pFrame->pNext){
    OutputVar(xsltFile, pFrame, sFrom);
}

//Iteratively output the XSLT variables for the iframes of the current var
for(pIFrame = pVar->pFirstIFrame; pIFrame ; pIFrame = pIFrame->pNext){
    OutputVar(xsltFile, pIFrame, sFrom);
}
```

FIG. 35



OutputChain

```

pElement = pChain->pFirstElement;

bChainBaseIsFound = false;

while(pElement){
    if(pElement->bIsChainBase) bChainBaseIsFound = true;

    if(pElement->Ele.iFamilyId != EMPTY_ELEMENT){

        szRoot = ((pElement->Ele.sPath.substringData(0, 3))
            + (pElement->Ele.sFrame)).transcode();
        <<xsl:apply-templates select="'szRoot[sPath]'"
            mode="'szRoot_output_ pElement->Ele.iFamilyId'"/>

        NEW_UNIT(pUnit, pElement);

        if(pElement->Ele.bIsChanged == true || pElement->Ele.sNewTag.length() != 0)
            //this Element is changed
            APPEND_UNIT(pCard, pUnit); //the Queue will be scanned later
        }
        else{
            APPEND_UNIT_TO_FAMILY(pCard, pUnit);
        }
    }
    else{ //it is the empty Element which is the content of the parent Element of this chain
        szRoot = ((pChain->pParentElement->Ele.sPath.substringData(0, 3))
            + (pChain->pParentElement->Ele.sFrame)).transcode();
        <<xsl:apply-templates select="**|text()" mode="'szRoot_test_
            pChain->pParentElement->Ele.iFamilyId'"/>
    }
    pElement = pElement->pNext;
} //end of while

//if the chain has chain base and the chain base is not in the chain (i.e. it is moved to other chains
or it is deleted), append the chain base to family according to it family id.
if(!bChainBaseIsFound && pChain->pChainBase){
    NEW_UNIT(pUnit, pChain->pChainBase);
    APPEND_UNIT_TO_FAMILY(pCard, pUnit);
}

```

FIG. 36

